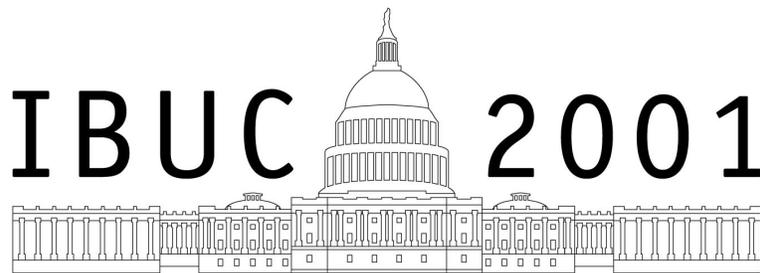


Proceedings of the 7th International Blaise Users Conference



September 12 - 14, 2001
Washington Marriott Hotel
Washington, DC
USA



Local Host Organization

Preface

The papers prepared for these proceedings were presented at the 7th International Blaise Users Conference held in Washington, DC, USA, on September 12 - 14, 2001. The conference included three days of technical papers and presentations on the use of Blaise and associated topics.

The conference program was planned and organized by the Scientific Committee of the International Blaise Users Group, chaired by Tony Manners from the Office for National Statistics in London. Other members of the committee included:

- Christophe Alviset (Institut National de la Statistique et des Etudes Economiques)
- Armin Braslins (Statistics Canada)
- Lon Hofman (Statistics Netherlands)
- Donna Jewell (Research Triangle Institute)
- Vesa Kuusela (Statistics Finland)
- Asa Manning (U. S. National Agricultural Statistics Service)
- Brett Martin (Statistics New Zealand)
- Bill Mockovak (U. S. Bureau of Labor Statistics)
- Sarah Nusser (Iowa State University)
- Mark Pierzchala (Westat)
- Ellen Soper (U. S. Bureau of the Census)
- Fred Wensing (Australian Bureau of Statistics)
- Gina-Qian Yang (University of Michigan)

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

IBUC  2001

7th International Blaise Users Conference Washington, DC September 12 - 14, 2001

Session 1: Major Themes in Blaise

- Meeting the Challenges of Managing CAI in a Distributed Organization
Asa Manning, National Agricultural Statistics Service, USA
- Revolutionary Paradigms of Blaise
Mark Pierzchala, Westat, USA
Tony Manners, Office for National Statistics, United Kingdom
- After 10 Years of CAPI, INSEE Embarks on the CAPI 3 Project
Mario Avi and Philippe Meunier, INSEE, France
- Challenges of Instrument Development
Ellen Soper and Ed Dyer, U. S. Bureau of the Census, USA

Session 2: Programming Techniques

- Dynamically Created Answer Categories in Blaise Instruments
Carol Oppenheim, Battelle, USA
- Different Ways of Displaying Non-Latin Character Sets in Blaise
Leif Bochis, Statistics Denmark
- Programming Techniques for Complex Surveys in Blaise
David Altvater, Ventrese Stanford, and the Blaise Authoring Team
U. S. Bureau of the Census Technology Management Office
- Output Processing for Consumer Expenditure Survey
Xiaodong Guan and Howard R. McGowan
U. S. Bureau of the Census

Session 3: Case Management I

- LAURA
Marko Sluga and Janez Repinc
Statistical Office of the Republic of Slovenia
- Interactive Shelling Between Visual Basic and Blaise
John Cashwell, Battelle, USA
- Standardised Survey Management at Statistics Netherlands: An Example of the Object Approach
Frans Kerssemakers, Marien Lina, and Lon Hofman
Statistics Netherlands`
- Survey of Labour and Income Dynamic (SLID) Project at Statistics Canada
Richard Chan, Statistics Canada

Session 4: Case Management II

- A Windows-Based User Interface and CAPI Information System
Vesa Kuusela, Statistics Finland
- Redesigning CAI Systems to Handle Multiple Authoring Languages
Michael Haas, U. S. Bureau of the Census
- The Annual Survey of Manufactures (ASM) at Statistics Canada
Okuka Lussamaki, Statistics Canada
- Case Management for Blaise Using Lotus Notes
Fred Wensing, Australian Bureau of Statistics
Brett Martin, Statistics New Zealand
- LWR: An Integrated Software System for the Household Expenditure Survey
Thomas Pricking,
Federal State Bureau of Data Processing and Statistics, Germany

Session 5: Data Editing

- A Blaise Editing System at Westat
Rick Dulaney and Boris Allan, Westat, USA
- Data Entry and Editing of Slovenian Agricultural Census
Pavle Kozjek, Statistical Office of the Republic of Slovenia
- The Role of Error Detection and Correction Systems in Survey Management and Optimisation
Elmar Wein, Federal Statistical Office, Germany

Session 6: Web / CASI

- Some Techniques for Internet Interviewing
Adriaan Hoogendoorn, Vrije Universiteit, Amsterdam, The Netherlands
- Audio-CASI with Challenging Respondents
Rebecca Gatward, Office for National Statistics, United Kingdom
- Dynamic ACASI in the Field: Managing All the Pieces
Boris Allan, Kathleen O'Reagan, and Bryan Lohr
Westat, USA
- Computer-assisted Self-interviewing over the Web: Criteria for Evaluating Survey Software with Reference to Blaise IS
John Flatley, Office for National Statistics, United Kingdom

Session 7: Blaise API

- Blaise API, A Practical Application
Andrew Tollington and Pauline Davis
Office for National Statistics, United Kingdom
- Some Examples Using Blaise Component Pack
Gina Qian Cheung, University of Michigan, USA
- Internet Interviewing Using Blaise API
Bas Weerman, CentERdata, University of Tilburg, The Netherlands

Session 8: Audit Trails and Testing

- Testing a Production Blaise CATI Instrument
Amanda F. Sardenburg, U. S. Bureau of the Census
- The Practical Application of Audit Trails: A Case Study
Rob Bumpstead, Office for National Statistics, United Kingdom
- Automated Testing of Blaise Questionnaires
Jay R. Levinsohn and Gilbert Rodriguez
Research Triangle Institute, USA
- Reporting on Item Times and Keystrokes from Blaise Audit Trails
Sue Ellen Hansen and Theresa Marvin
University of Michigan, USA

Session 9: Blaise CATI

- Integrating the Use of Data Centers into the NASS CAI Structure
Roger Schou, National Agricultural Statistics Service, USA
- Overview of Blaise at Iowa State University Statistical Laboratory
L. L. Anderson, J. M. Larson, D. G. Anderson, S. M. Nusser
Iowa State University, USA
- Scheduler Issues in the Transition to Blaise
Leonard Hart, Mathematica Policy Research, Inc., USA
- Considerations in Maintaining a Production Blaise CATI System
Kenneth P. Stulik, U. S. Bureau of the Census

Session 10: Experience and Organization

- Organisation of an Extensive Blaise Project
Holger Hagenhuth, Federal Statistical Office, Germany
- Welfare Benchmark 'Easy to Use Local/Internet DEP' for the Dutch Municipal Government Officials
Carlo Vreugde, VNG, The Netherlands
- Blaise on Pentops, Blaise MetaData and Other Blaise Experience in the National Health and Nutrition Examination Survey
David Hill and Christina Kass, Westat, USA
Debra Reed-Gillette and Lewis Berman,
Centers for Disease Control and Prevention / National Center for
Health Statistics, USA

Paper Not Presented at Conference

- Blaise to OLE-DB-Relational Data Migration? Yes!
T. R. Jellema, NAMES B.V.

Session 1: Major Themes in Blaise

- Meeting the Challenges of Managing CAI in a Distributed Organization
Asa Manning, National Agricultural Statistics Service, USA
- Revolutionary Paradigms of Blaise
Mark Pierzchala, Westat, USA
Tony Manners, Office for National Statistics, United Kingdom
- After 10 Years of CAPI, INSEE Embarks on the CAPI 3 Project
Mario Avi and Philippe Meunier, INSEE, France
- Challenges of Instrument Development
Ellen Soper and Ed Dyer, U. S. Bureau of the Census, USA

Meeting the Challenges of Managing CAI in a Distributed Organization

Asa Manning, National Agricultural Statistics Service, USA

Standardizing procedures are an important part of any statistical organization's attempt to reduce non-sampling errors. The physical structure of the National Agricultural Statistics Service (NASS) with its 45 field offices distributed across the county makes standardization even more important. For those supporting the CASIC activities in that environment, enforcing standards becomes an absolute necessity. Without the standards, support would be impossible.

NASS Structure

It is important to understand the basic structure under which NASS operates. NASS has approximately 400 employees in its headquarters located in Washington, DC and in nearby Fairfax, VA. The remaining 750 NASS employees are located in the 45 field offices, which NASS calls state statistical offices (SSO's). National survey activities are distributed between HQ and the SSO's as follows:

<u>HQ</u>	<u>SSO's</u>
Instruction manuals & training	Interviewer training
Sampling	<i>Data collection & editing</i>
<i>Instrument development (Paper and Blaise)</i>	Survey administration
Coordination	Analysis
Compilation of national estimates	State estimates

CASIC Implementation at NASS

Blaise is used in 43 offices for both data collection (CATI) and interactive editing. The following table summarizes the Blaise 4 applications that NASS currently uses:

Blaise 4 Applications In Use At NASS

Survey Application	Frequency	No. of States	Type	Peak CATI Volume	Peak Editing Volume	Post-Blaise Processing
March Crops/Stocks	March	42	Multiple Frame	25000	50000	mainframe
June Crops/Stocks	June	42	Multiple Frame	25000	50000	mainframe
September Crops/Stocks	September	42	Multiple Frame	25000	50000	mainframe
December Crops/Stocks	December	42	Multiple Frame	25000	50000	mainframe
Cattle Report	January & July	43	Multiple Frame	20000	40000	mainframe
Hog Report	Monthly	19 to 43	Multiple Frame	5000	16000	mainframe
Sheep Report	Jan. & July	42	Multiple Frame	7500	15000	mainframe
Agricultural Labor	Jan., Apr., July, & Oct.	42	Multiple Frame	6000	12500	mainframe
ARMS/Chem. Use Screening	May - July	42	Multiple Frame	45000	50000	mainframe
Cotton Gins	Aug. - Apr.	13	List Frame	1000	1000	LAN
Cattle on Feed	Monthly	13	List Frame	1000	2000	LAN
Catfish	Jan. & July	15	List Frame	1000	1400	LAN
Trout	January	17	List Frame	250	450	LAN
Chicken & Egg	Monthly	30	List Frame	500	1000	LAN
Bee & Honey	December	42	List Frame	2000	5000	LAN
Ag. Yield	May-Nov	42	List Frame	35000	37000	mainframe
June Area	June	42	Area Frame	N/A	200,000	mainframe
Pest Database	Continuous	42	Special	N/A	N/A	N/A
Reimbursable	Varies	Varies	List Frame	Varies	Varies	Varies
Census	Every 5 th year	42	Census	200000	N/A	Unix/main

In NASS, our Blaise developers and end users are completely separated. All of our national applications are developed in HQ, but no data is collected or edited there. That activity is located in the SSO's. Not

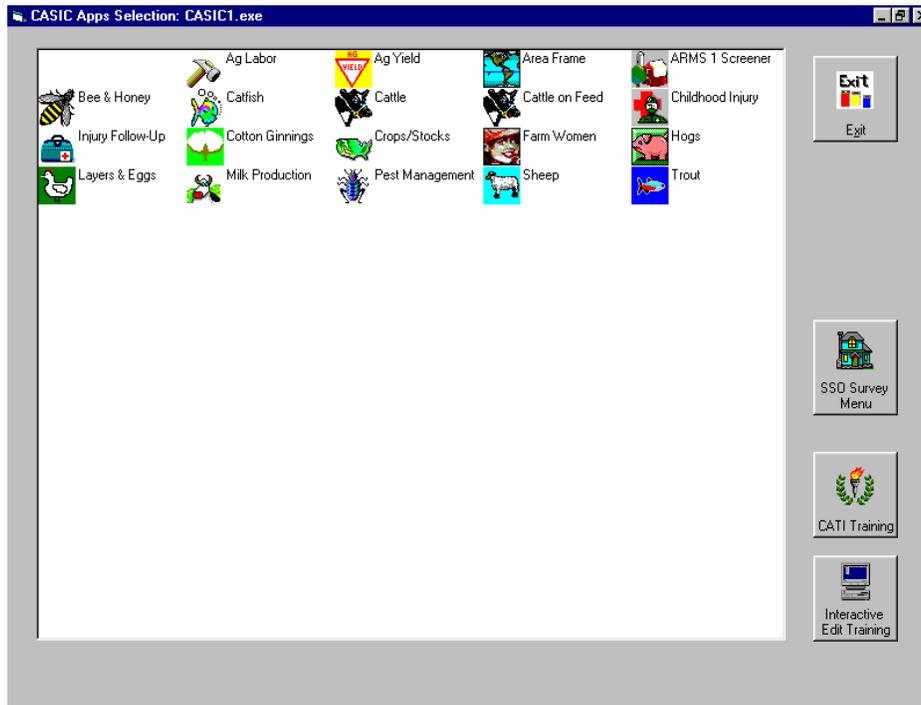
being able to communicate face to face with our end users also presents some challenges. Depending on telephone contacts to resolve problems is very inefficient and frustrating. Standards help reduce the need for support and also serve to expedite it when it does occur.

All of the above applications are developed and supported by the CASIC and Editing Section in HQ. Currently there are only five people that handle this activity. During the busiest time of year each SSO will have from 5-10 Blaise applications running concurrently. Without a stable hardware and software environment, the support demand could easily overwhelm the HQ support staff. NASS LAN policy assures that we will find almost identical LANs in every SSO. Stability in the Blaise software is obviously a must. We will often use a new version of Blaise in HQ for several months before deploying it to the SSO's. This provides a chance to shake down the new version before it is used in production.

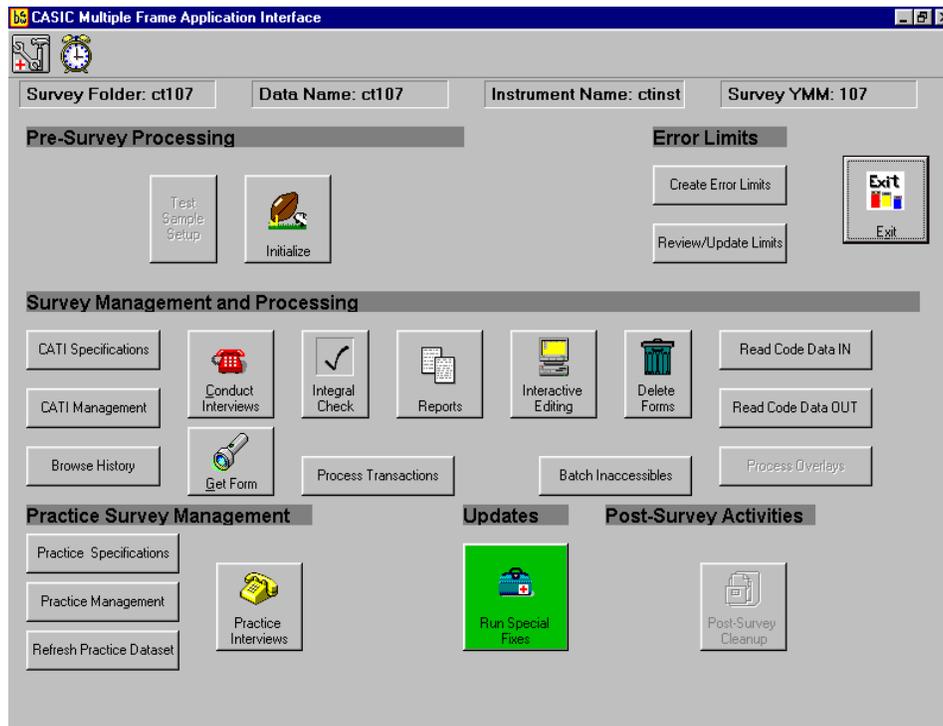
Since staff in the SSO's may be working with multiple Blaise applications simultaneously, we take care to make certain that where possible those applications look and feel the same. Strictly enforced CASIC standards assure that there is consistency from application to application and state to state. Several techniques that we use will now be discussed.

Standard Interfaces

When NASS moved from Blaise III to Blaise 4, we developed a series of standard interfaces, which were developed in Visual Basic, for our users in the SSO's. When the staff in the SSO's want to start a Blaise application, they click on an icon on their desktops. The following screen appears.



This screen has an icon for every Blaise 4 application that is available. From this screen the user clicks on the desired application. The standard CASIC interface for managing any Blaise application then appears.



This interface is identical for every Blaise application. It is organized into separate areas which correspond to phases of the survey process, such as pre-survey processing, survey management and processing, updates, and post-survey activities. These areas of the screen never change from application to application. Once a button is programmed into a position on the interface, no other button will ever use that position. This explains the empty area to the left of the “Test Sample Setup” button on the interface. The Crops/Stocks application instruments are prepared in the SSO, so a “Prepare Instrument” button appears on the interface for that application. Since HQ prepares the instrument for Cattle that position remains blank on the interface in our example. Buttons are present but inactive if certain key input files are not available. The “Test Sample Setup” button is an example. The general flow of the processes is from left to right. The Survey Management and Processing moves from CATI management on the far left, then CATI, then Interactive Editing. The bright green color of the “Run Special Fixes” button is designed to attract attention to the need to run an update. This indicates that the CASIC & Editing Section has delivered a fix for this application and it has not yet been applied. This button will be the same color as the other buttons and disabled once the fix has been applied. All of these functions make it easier for the end user in the SSO’s to manage their Blaise application without HQ assistance.

Standard Blaise Shells

When the CASIC & Editing Section develops a Blaise application, we only have to program the questions specific to the application. NASS already has a series of shells that handle all of the administrative type questions. The developer essentially “wraps” the appropriate shell around the code they developed by including the pre-existing shell code in the preparation process. Once prepared, the instrument automatically meshes with existing Manipula processes and with the CASIC interfaces just discussed. This assures that administrative questions on similar Blaise surveys always function the same. One key ingredient in the success of this approach is that NASS has already committed to the use of

standards in all survey instruments, paper and automated. NASS uses only four types of administrative designs on its questionnaires. They are multiple frame, list only, census, and area. Blaise shells are maintained for each one. The initial effort to build these shells was significant. Maintaining them requires some time, but because the original effort was designed to be flexible, is actually minimal. The amount of development time saved in each application has paid back the shell development time many times over.

Application Distribution

NASS uses a standard method called LAN Updates to deliver any software related update to the SSO's. SSO's will receive about 500 updates each year with CASIC related updates accounting for about half. All LAN Updates are coordinated by staff within the Information and Technology Division (ITD). The staff in the CASIC and Editing Section prepares the LAN Update. It will consist of a series of zip files which correspond to destination directories on the SSO LAN, an install BAT file which will unzip the files onto the SSO server, and documentation. The Blaise developer in HQ will login to a HQ server which is configured like the SSO servers. The install process is then tested, followed by application testing once successfully installed. Once all testing is completed, the LAN Update is forwarded to ITD. They assign the LAN Update a unique number and place it on a server in HQ. The SSO is then notified by email that the LAN Update is available. The SSO then must use FTP to move the LAN Update onto their server. They then follow the instructions to install the update. By using the rigid policy to manage software updates going to the SSO's, NASS protects the SSO's from poorly planned updates. It takes extra time in HQ to prepare the LAN Update, but if efficiency is gained in 43 receiving offices, the time in HQ is worthwhile.

The CASIC and Editing Section can build certain safeguards into the installation batch file. We may look for certain files to assure that the application being updated is present or current. We may also look for the presence of users that may already be in the Blaise application. The presence of these safeguards are designed to avoid problems created by prior updates being missed or files being open when we try and replace them. Even though the SSO's are given specific instructions that should assure these things don't happen, you can't count on all instructions being followed exactly in 43 offices every time.

NASS strives very hard to avoid changing the data model after data collection has begun. It is very problematic to coordinate a data model change in our distributed environment. Thus, unless the problem being fixed is significant, we will usually find an alternative way to deal with it.

Using ReachOut and the WAN to Aid Support

Solving a problem that surfaces in an SSO can be difficult with the best of tools. Listening to someone describe a situation over the phone without being able to see their workstation wastes tremendous amounts of time. One of the biggest aids for support at NASS in the last few years is the availability of our Wide Area Network and a software package called ReachOut. Staff in the CASIC and Editing Section can now "ping" a workstation in an SSO and watch as applications run. The savings in support time are not easy to measure, but we would estimate a savings of 75 percent.

Security is controlled by the SSO. We can only reach the other workstation if the office has set up ReachOut on it and gives us the necessary ReachOut login and password. The SSO can also give control of the workstation session to the HQ user when necessary to expedite finding a solution.

CASIC Coordinator Training

Probably the most significant factor that determines the amount of support demand from an SSO is the level of expertise available in that office. Most support calls do not really require a developer, but rather an experienced Blaise user. When we are able to elevate the Blaise knowledge in an office, we see a reduction in support calls from that office. Since NASS adopted Blaise as our primary CASIC software in the early 1990's, we have twice held a series of CASIC Coordinator workshops. These workshops were designed to teach SSO staff how to manage the CASIC process, not develop applications. The 3.5-4.5 day agenda consisted of sessions on CATI Management, using data collection and editing instruments, using Blaise utilities to monitor and solve problems, and generally how to manage the flow of data through the Blaise process.

We held these workshops prior to introducing Blaise III and Blaise 4. We trained two staff from each SSO during the first series of workshops and one in the second. Since the change from Blaise III to Blaise 4 was less dramatic than the original shift from CASES, we felt that one well-trained person in each office would be sufficient. Once the person returned to the SSO from the workshop, it was their responsibility to help manage the first Blaise 4 applications that arrived in the office. They were also charged with passing their expertise to others in the office. Almost everyone in the office would touch Blaise, so it was important for this knowledge to be shared. The success of this training transfer depends on the individual's training skills and the commitment of management in the office to allot time for that training. Since we can't train everyone in each SSO, training an "expert" and depending on the transfer of that expertise to others is at the core of all NASS training.

Blaise Developer's Training

Each SSO has certain survey applications which are not covered by the HQ survey umbrella. For these applications, the SSO must develop their own survey processes. Since Blaise is the standard tool for CATI and IE, most SSO's need the ability to develop their own Blaise applications. To support this need, the CASIC and Editing Section developed a training course for potential SSO developers. This 3.5-day course consists of a series of slide shows about the Blaise language, followed by a series of hands-on exercises. During the exercises, each participant gradually assembles pieces of a crop acreage and production instrument. The course closes with an introduction to Manipula and guidelines on using a shell designed for state applications. If the shell is used, the developer also benefits from being able to easily add their application to the standard CASIC interfaces without knowledge of Visual Basic.

NASS has conducted six of these workshops over the last seven years. Thirty-two states have developed their own applications. These workshops and subsequent observations of the participants as they develop their own applications provide us with a glimpse of potential candidates for the CASIC and Editing Section. We don't advertise it, but the developer workshop is an audition of sorts.

One side benefit to an SSO having a developer in house, is a reduction in support needs in that office. That developer has an understanding of Blaise that exceeds almost every CASIC Coordinator. That understanding translates into a reduction in support calls from that office, because the local expertise has been elevated.

Application Testing

In an environment where support is challenging, obviously the easiest way to minimize the challenge is to eliminate the need for support. Perfect applications require almost no support. We have not achieved

perfection, but do as much testing as possible before applications go to the SSO's. We have up to four phases of testing depending on the complexity and stability of the application; developer testing, special testing in HQ by SSO staff, testing of a preliminary instrument in the SSO's, and a review of the final application.

Simpler applications that have changed only slightly will be tested by the developer and then the SSO's will test the application once the "final" application is received. No rigorous testing in the SSO's takes place.

For the more complex applications, we add a phase where the SSO's test a preliminary instrument. This instrument is delivered to the SSO's about 3-4 weeks prior to the start of the survey. The preliminary instrument is distinguished from final instruments by a bright green background to minimize the likelihood that it will mistakenly be used for live data collection. The SSO's run cases completely through the preliminary application to test the entire process. They communicate any problems to the CASIC and Editing Section and those bugs are removed before the final instrument is delivered to the SSO's. The final application is due in the SSO's 3-5 business days before the start of the survey.

The Crops/Stocks application is even more challenging in that every SSO has a unique instrument. Because of this additional complexity, we will at times include a special phase of testing as the HQ developer begins to create the preliminary instruments. We will bring one of our strongest CASIC coordinators from an SSO into HQ to test each individual state's instrument. They will help update the Blaise specifications database that is used to assemble the instruments, generate the state instruments, and verify that each one is accurate. Once that phase is complete, the preliminary instrument is delivered to the SSO's and testing of the preliminary instruments continues as described in the previous paragraph.

Challenging Instrument Requirements

NASS maintains three applications that present special challenges. The Crops/Stocks CATI/IE and June Area IE applications have unique instruments for every state. The Census of Agriculture CATI instrument has 14 unique regions. All of these surveys collect information on crop acreage and production. Since crops vary by state or region, so do the instruments. NASS employs three different methods of developing these instruments. All of these have been the subject of entire papers at previous International Blaise User's Conferences, so only an overview will be provided here.

Crops/Stocks This application collects information on crops grown and grain stored in March, June, September, and December. The initial efforts on Crops/Stocks go back to Blaise 2.

First, we created a library of modules for every combination of questions ever asked for each crop. For example, early in the growing season only planted acres might be asked, while in December we might ask acres planted, harvested, and grain produced. The same questions are asked for multiple crops, so we built a structure for each combination of questions. Then we generated the actual crop specific code module by feeding the individual crops into the structure using Manipula. By generating the crop modules from a single clean structure module, we assured consistently clean crops modules. This step was called the code generator.

We maintain a Blaise specifications instrument with a record for each state and quarter. The Blaise instrument brings up a series of questions on each crop and stock item. The user indicates whether the commodity will be included, and if so what questions will be

asked. Once the information has been input, we use Manipula to read the specifications and assemble the needed code modules. Then through a series of batch files, the instrument is concatenated and prepared. We call this process the instrument generator.

This process performs the customization at preparation time. The entire library is distributed to the SSO's and instrument is prepared there. This approach has allowed us to reduce the staff time needed to create the 42 unique instruments and at the same time produce an instrument that is more reliable.

June Area This application is used to edit data that is collected via personal interviews with farmers each June. Essentially, the major purpose of the survey is to identify the crops grown on a tract of land. As with the Crops/Stocks application, the crops for each state are unique. The IE instrument was originally developed as part of a CAPI research project. The IE instrument was kept after the CAPI effort was discontinued. A different method is used for handling the unique crops in this instrument.

We build a toggles file for each state that lists the crops to be edited. This file is read at run time to customize the crop section for the state in question. This application is now one of the most popular Blaise instruments used in NASS because of the tremendous time savings gained by editing this data interactively rather than with the old batch edit.

Census The Census of Agriculture is conducted every five years. The Blaise instrument is only used for CATI. No editing is done in Blaise. The instrument must be able to collect any crop, no matter how minor, grown anywhere in the United States. Our typical survey instruments only collect data for the crops that are included in our estimation program. Many minor crops are not included. On surveys, we ask the farmer about each crop. That method would not work on the census because the list of crops is so vast, so we used a completely different technique.

Before we ask specific acreage questions, we build a profile of crops grown on the operation. We ask the farmer to identify the crops grown and one by one capture them using a trigram coding module. When the farmer indicates that all crops have been covered, the interviewer reviews the list of crops with the farmer to make sure nothing has been overlooked. Special efforts are made to screen for certain crops such as hay, which experience has taught us farmers tend to forget. Once the list is complete, we ask the appropriate acreage, production, and irrigation questions for each crop identified in the profile.

This was the first time we used the trigram coding module at NASS. This technique was very successful and we plan on using a similar process during the 2002 Census of Agriculture.

Summary

As you can see, NASS uses a variety of techniques to attempt to minimize the support demands for our Blaise applications. We feel that these standardization techniques, which we have implemented over a period of over eight years, are the only reasons that we are able to support our many applications with a small number of developers. The NASS structure gives us very little choice. We hope other organizations can learn from our experience. Even if you don't need to enforce standards as rigidly as we do, some of these techniques might serve your organization as well.

Revolutionary Paradigms of Blaise

Mark Pierzchala, Blaise Services at Westat, United States
Tony Manners, Office for National Statistics, United Kingdom

Abstract

Blaise has historically featured several revolutionary paradigms that make it one of the leading computer-assisted interviewing (CAI) systems. By proper appreciation and application of these paradigms Blaise can dramatically improve the way an organisation executes surveys.

Some examples of this revolution from current users illustrate how Blaise has enabled fresh approaches to tricky challenges. These examples cover a range of issues including interviewer usability, integration of data collection with post-collection processing, development of instrumentation, and data and metadata export and manipulation. With proper appreciation and application of these paradigms it is possible to rethink the CAI process and to achieve better data quality in a cost-effective way.

The paper will speculate on how a new paradigm, that of Open Blaise Architecture, continues the pattern of Blaise contributions to improved CAI processes.

Seven Major Paradigms

We first identify 6 aspects of the system that Blaise has embodied since its first versions. While there have been enhancement and elaboration of these paradigms over the years, it is argued that the choices made in its early years have resulted in Blaise's current-day success and worldwide adoption through its adaptability and capability in handling very difficult surveys. These 6 paradigms are *intuitive usability*, *total checking and total reliability*, *structured modularity*, *development based on researcher skills*, *all metadata defined once*, and *no data without metadata*. We identify a seventh paradigm, the recently arrived *Windows openness*, and we speculate on its eventual impact.

Examples given below demonstrate how these revolutionary paradigms have enabled, and how they encourage, efficient ways of approaching all aspects of CAI and survey processing. Appreciation and use of these paradigms are cost effective and have improved data collection and processing.

In this paper, a paradigm may be deemed revolutionary if it represents an approach to a challenge that is a significant departure from previous practice. The discourse identifies how each of the paradigms is a departure from approaches taken in most other systems.

While these seven paradigms are extremely beneficial, there will always be challenges that may require senior and experienced staff to design and develop Blaise instruments along with other non-Blaise expertise and systems. This is particularly true for some very large and complex studies, studies with special requirements, or when shifting an existing survey from another CAI system into Blaise.

Intuitive usability

The default Blaise split-screen display is a readily recognizable hallmark of the system. It combines a question-display area (top part of the screen) with a page display (the bottom part). The question text that is displayed at any moment corresponds to the location of the cursor in the page.

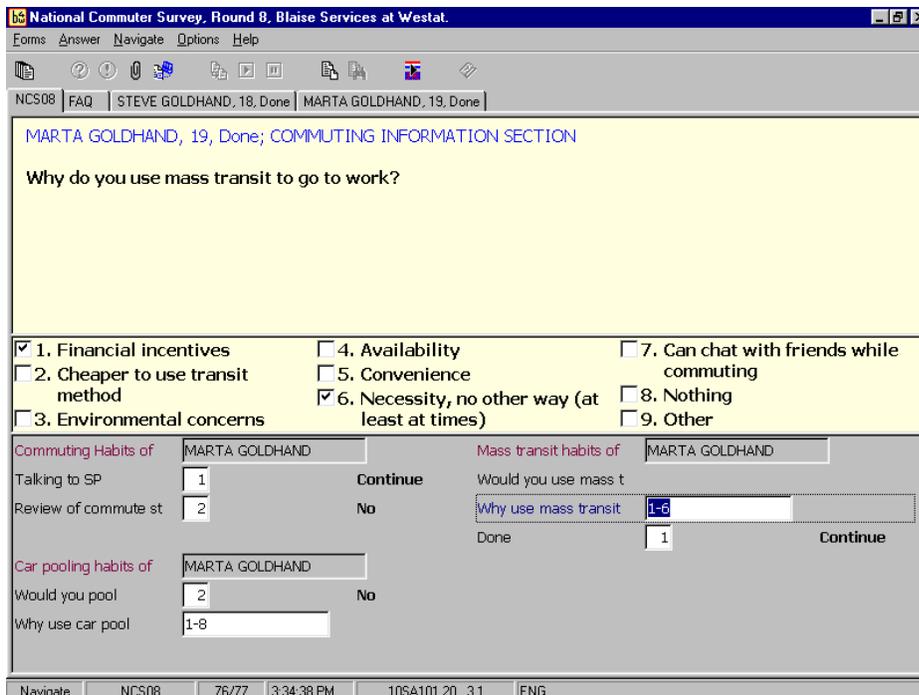


Figure 1: An illustration of the split-screen interface with a formatted Blaise page.

The page in the bottom part of the screen is the basis for the proven Blaise usability for interviewers¹. By default it is a multi-question display that tells an interviewer about a cluster of related questions, showing which have been answered (with values), which were not on route and which are still to be answered. This information is vital to an interviewer's sense of being in control of interview and combines well with the literal control the page gives for navigation. The Blaise instrument page has many similarities to a page in a paper questionnaire and can contain labels, white space, (multi-lingual) field descriptions, texture, page numbers, and other attributes that enable the interviewer to recognize a page's topic and place in the instrument. The page, if designed well, makes a meaningful element, which enables the interviewer to build an instant mental overview of the questionnaire and where they are in it at any time. Interviewer briefing for the survey need only show the interviewers the high-level structure of the instrument in terms of topic order, and the page can indicate the current topic. The page is also a unit of navigation with the Page Up and Page Down keys, or you can navigate within a page using arrow keys or a mouse. There are many ways to navigate in a Blaise instrument but the page offers interviewers a method that is transparent for them and retains their control of navigation. It is particularly popular for backing up over short distances in the instrument (by far the most common need for backing up which is not triggered by an edit check which has its own direct movement to the variable to be changed). The higher the average data density in an instrument's pages the fewer overall pages that are needed. Twelve intuitive methods of navigation are documented in Pierzchala and Farrant (2000). As we have stressed, the Blaise page gives the interviewer the secure sense of being in control of the technical aspects of the CAI interview, something that is important for the overall quality of interviewing. It should be noted however, that there may be methodological or other reasons to limit navigation: Blaise can accommodate this too.

¹ Interviewer choice of Blaise as the most interviewer-friendly system was a key aspect in its selection in a competitive tender for CAI software for a major British survey in the early days of CAI when such comparisons could be made with interviewers who had not previously been exposed to any of the systems.

Revolutionary aspect: The Blaise multi-question page stands in stark contrast to the default display convention of most other CAI systems that are oriented towards an item based display of one question at a time. The item-based display can result in the well-known *segmentation effect* (Groves, Berry, and Mathiowetz 1980) where interviewers have little idea of the location or context of questions as they conduct the interview. It is very difficult for them to see the relationships of questions and groups of questions to one another. In these other systems navigation and error repair is difficult even if the system architecture supports backup. The well-designed Blaise page helps eliminate the segmentation effect.

Benefits may include reduced training costs, easier ad hoc navigation and data correction during the interview, easier re-entry into the system, and lower post-collection processing effort. Training costs may be reduced because less time is spent teaching the questionnaire to the interviewer. They can see the whole questionnaire in much the same way as an interviewer can see a paper questionnaire. It is easier and more natural for interviewers to learn the instrument from the meaningful topics in pages than it is from items in item-based systems.

In Europe, where Blaise has predominated for over a decade, interviewers are expected to navigate and make data corrections. Thus instead of getting a note that someone must review post-interview, you should expect to get a corrected value instead, and if necessary, any data from a newly established route. (However, there may be methodological reasons to limit navigation back to certain areas of the instrument. In this case you may still get notes that need to be reviewed.) Clean data from the field are the default expectation in an organisation like the United Kingdom's Office for National Statistics (ONS) which expects interviewers to edit during the interview - indeed, which sees this as a key advantage of CAI. More than a decade's experience of using Blaise has shown that it can be achieved reliably.

During re-entry into the system, it is possible for the interviewer to review data previously entered to get an idea of what the interview is all about. Lower post-collection costs come about in three ways. One is fewer errors and notes to be reviewed post collection. Second is that the page-based overview is also available and beneficial to the data editor. Third is that Blaise has powerful edit and note-review facilities (see below).

Examples of U.S. instruments that take full advantage of the page-based usability include the Consumer Expenditure Survey (U.S. Bureau of Labor Statistics/U.S. Bureau Of the Census, Stanford, Altwater, and Ziesing, 2001), the June Area Frame Survey (U.S. National Agricultural Statistics Service, Pierzchala, 1996), and the Spanish Bladder Cancer Survey (U.S. National Cancer Institute/Westat, Frey 2000). All ONS surveys use the page concept: recent examples are the Expenditure and Food Survey (Office for National Statistics, Gatenby 2001), and Survey of Psychiatric Morbidity among Adults 2000 (Office for National Statistics, Singleton 2001).

Total checking and total reliability

Every time a new or changed data value is entered, Blaise re-executes the rules of every block affected by the new value. That is, it re-evaluates flow, re-computes assignments, and re-assesses the edits within all affected blocks. If appropriate, the routing of the instrument is changed or an edit is invoked immediately. This continual and global re-checking guarantees the integrity of data relationships in the questionnaire no matter where in the instrument the user makes changes. Total checking and total reliability is manifest whether in data collection mode or in post-collection editing mode.

Revolutionary aspect: The continual re-checking of appropriate blocks is in contrast to limited or no rechecking seen in other systems. When the re-checking is started, Blaise makes no assumption about the correct route of the interview. Rather it determines anew the correct route every time there is a new value. The states of the instrument are constantly re-assessed and are re-established every time a new data value

is entered or the form is brought into memory. A variant on this is passive checking in edit mode where the data editor can delay the checking until a function key is pressed, allowing several values to be changed before the next re-assessment of the form.

The constant re-checking is one of the architectural enablers of the 12 methods of navigation. The interviewer can navigate and make corrections with confidence that the constant global re-checking will ensure the correctness of the route and edits. This paradigm also allows one instrument to handle both interviewing and editing mode, even where editing is on data from paper questionnaires. An example of this is from the Quarterly Agricultural Survey (NASS) where every quarter half the cases are collected in Blaise CATI and half are done in the field on paper. The same Blaise instrument handles both the CATI and the data editing no matter where the data originated (Schou, 1995). Thus post-processing is a natural capability of the Blaise instrument, eliminating the need to rewrite all data relationships in another system to handle post-collection review.

Other examples of multi-mode use include the UK's CAPI/CATI rotating panel Labour Force Survey (ONS/Manners 1992, Elliot 2000), and Early Childhood Longitudinal Survey-Kindergarten cohort (U.S. National Center for Educational Statistics/Westat, Dulaney and Allan, 2001).

Structured modularity

The Blaise development language has several modular constructs including types (pre-defined pre-codes), procedures, blocks, and tables. The block is the most important and most common of these modular constructs. All of these modular constructs can be used or reused in different parts of the same instrument or in different instruments. Blocks and other modular constructs model natural questionnaire structures, such as a section of a questionnaire, a table, or a row within a table.

An instrument may be considered to be a collection of blocks and other modules and this modularity offers maximum order and a maximum degree of flexibility. It is possible to parameterize a block and use it in a variety of different circumstances. Specification and development can be block-based, independent of the context of the whole instrument. Instrument development can proceed on two levels, that of constructing blocks (and other modules) and that of tying them together.

Revolutionary aspect: Specification and development can be structure-based as opposed to an item-based approach. You can build a library of types, procedures, blocks, and tables, and combine these with project-specific modules. The ability to re-use blocks and other modules encourages and enables the creation of organisation standards and standard survey management modules. It is even possible to define instrument architecture for a program of surveys and drop survey specific blocks into that pre-defined organization (Manners, 1998), (Pierzchala and Manners, 1996). An additional advantage of pre-developing blocks and modules and using them across surveys is in the way this enables data to be compared across surveys (Manners ET al, 2001). This modular aspect of instrument specification and development is used to great effect in NASS where every quarter 44 instrument versions of the Quarterly Agricultural Survey are generated (Pierzchala, 1992), and where one June Area Frame instrument is driven 44 different ways by an on-line specification (Pierzchala, 1995 and 1996).

Methods of handling variations on a theme have been developed in NASS (Pierzchala, 1992 and 1996) and in the U.S. Bureau of Labor Statistic's Consumer Expenditure Survey (Stanford, Altvater, and Ziesing, 2001). There, many so-called 'laundry-list' tables share common table-level specification and source code files. The row-level specifications and source code differ but are driven to a great extent by an on-line specification database, while much of the rest of the row-level sources follow agreed-upon model code.

ONS also makes extensive use of standard high-level block definitions such as table-level constructs, and also of model code. This allows the researcher to focus on the subject matter and block-level details without having to worry about how these are all tied together at the instrument level (Manners, 1998). For example, the basic structure of the Expenditure and Food Survey (the interview element of which has a mean length of 90 minutes) is a set of Blaise tables in which each member of the household has a constant position (fixed by the first respondent's original choice of order in listing them). Household members can actually be interviewed within this structure in any order (and in differing orders in differing blocks, though this would be an unusual choice) and on any number of different occasions, to suit availability. Such flexibility is not unusual, but the important point here is that it is achieved with minimal effort. Blaise handles all the overhead of keeping track of who is being interviewed when - no additional programming is required. The effectiveness of this structure has been proved on the UK's Family Expenditure Survey (the forerunner to the Expenditure and Food Survey) since 1994.

Development based on researcher skills

Since it is possible in Blaise to divorce block-level development from higher-level instrument integration, it is also possible to have researchers program the blocks themselves. If you can do SPSS re-coding tasks you can do basic Blaise (Manners, 1998). This includes field definition, flow, edits, and computations. The existence of standard organisation-level blocks and programming and display standards relieve the researcher from having to worry about anything except for the subject matter at hand. This has been put to use at ONS (Manners, 1998) and NASS where subject-matter specialists do the subject matter related instrumentation directly.

Revolutionary aspect: This style of instrument development stands in contrast to computer programmers working from specifications. If properly implemented, and in the right circumstances, this can save time and effort and reduce communication problems. ONS, in particular, argues that it improves quality because the researchers are able to design directly in the instrument medium, as they used to do when the medium was paper. Among the important lessons that researchers learn by this method is how differently they should think about designing CAI instruments from the ways they thought about paper instruments. This method of instrument development is best implemented within a program of related surveys, for example agricultural production surveys in NASS (Schou 1995). Specialty applications, difficult and new requirements, and instrument-level block integration are best saved for researchers who are highly experienced in Blaise, or for true computer programmers.

All metadata defined once

It is possible, methodologically desirable, and most efficient when as much metadata as possible is defined in the Blaise instrument. This includes metadata used for data collection, post-collection review and data editing, exporting of data and metadata, and integration of Blaise instruments into an overall infrastructure. Blaise has several features that make it possible to handle metadata depending on mode of processing. These include changes in display, changes in instrument behaviour, and the ability to include mode in IF conditions within the instrument. Additionally Blaise allows you to define much descriptive metadata including metadata about export that might fundamentally recast the structure of the data into an analytical form quite different from the structure of the data needed for data collection (Pierzchala and Farrant, 2000). The additional Blaise capability of metadata manipulation allows you to generate customized data export routines and downstream data definitions from the metadata in the instrument.

Thus it is possible to accomplish multiple modes of data collection (e.g., CATI, CAPI) *and* data editing with the same system, *and* export data including metadata in any way you need them, all from one metadata specification.

Revolutionary aspect: The ability to accomplish data collection and data review and editing in the same system eliminates the need to reprogram the same metadata in a different system for post-collection processing for the purpose of cleaning data (however, there may be other reasons to include some of the same metadata in a downstream system, but that can usually be exported too). NASS uses the same instruments for both data collection and data editing in the Quarterly Agricultural Survey with the additional complication that about half of the cases for data editing are collected on paper questionnaires by field interviewers while the other half are collected in CATI. This is true despite the fact that there are fundamental differences between CAI and paper data collection in NASS (Pierzchala, 1996). The statement of both data collection metadata and data editing metadata in one system also promotes a more systematic consideration of what the differences between the two modes should mean. If data collection and data editing are separated into two systems, it is often the case that unintended differences in handling of the data between the two processes often creeps in and may go unnoticed. The Early Childhood Longitudinal Survey-Kindergarten cohort (NCES/Westat, Dulaney and Allan, 2001) and the UK's Family Expenditure Survey (Manners, 1993) offers another example of the utility of using one system for both tasks, despite some fundamental differences.

No data without metadata

When you define the blocks and fields of a Blaise datamodel, you define the data layout at the same time. This means that the data and metadata of an instrument are very tightly bound, and in fact cannot be separated in development. This relieves the developer from having to worry too much about the details of database organisation. Consequently the developer can concentrate on the subject matter of the study. The developer or instrument designer do have to be cognizant of data organization issues and export considerations (Pierzchala and Farrant, 2000). The tight binding between data and metadata also allows robust iterative development of an instrument. As long as old metadata and data files are both archived, it is very easy to update the database of an instrument to reflect a new data definition.

Revolutionary aspects: Instrument development and database layout are taken care of in the same development step. You don't have to worry about synchronising two disparate steps of the process. Thus instrument development can proceed more rapidly. Additionally, all necessary information about data description and other metadata are in one place. A particularly dramatic example of the benefit of this paradigm is in the NASS Quarterly Agricultural Survey where 44 thematically related but considerably different instruments are generated in batch, every quarter. The compilation of each instrument results automatically in the creation of a unique database definition. The population of the database survey management data and other data is also done automatically in batch based on generated links between each database and the agency data infrastructure.

Windows openness

Windows openness is a new Blaise capability that works through COM and Active X components. This allows you to integrate Blaise with other systems. The Blaise API (Application Programmer's Interface) allows access to data and metadata from another system such as Microsoft Visual Basic. You can also build external file lookups using a standard relational database system as the external file. And you can call the Blaise data entry system as a DLL from another system.

Revolutionary aspects: The Blaise openness gives you the ability to add functionality within the context of an established CAI system, making use of the years of experience and testing which are built into it. Blaise will never be able to give you everything you need for all surveys. Windows openness allows you to develop special capabilities that Blaise does not yet have or may never have.

We see the first production applications of this capability in survey management tools. For example, a survey management shell could be written in Visual Basic with survey management data held in an Access database, but with direct access from the survey management shell to the Blaise data and metadata. A second kind of application will be in the more flexible extraction of data and metadata. Instead of exporting data en-masse, it will be possible to select fields and set up a custom export from a VB selection application, complete with qualifying metadata. Another application could be the development of special data entry programs, for example for special data collection needs not supported by traditional Blaise DEP.

But it is now also possible for third party developers to create value-added tools that work on the periphery of the core Blaise system. These might be commercial endeavors, the development of specialised or proprietary capabilities, or more informal trading of specially developed tools.

References

Dulaney, R. and Allan B. (2001). A Blaise Editing System at Westat. *Proceedings of the 7th International Blaise Users Conference 2001*, Washington, D.C.: Westat.

Elliot, D. (2001). Changes to the design of the Labour Force Survey. *Survey Methodology Bulletin no.47 July 2000*. Note: from no.48 (see reference to Gatenby below) the bulletin changed its name slightly with the addition of "Social".

Frey, R. (2000). Developing a Blaise Instrument for the Spanish Bladder Cancer Survey. *Proceedings of the 6th International Blaise Users Conference 2000*, Kinsale, Ireland. Central Statistical Office, Cork.

Gatenby, R. (2001) Overview of the development of the Expenditure and Food Survey (EFS). *Social Survey Methodology Bulletin, no.48, January 2001*, ONS, London

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.

Manners, T., Murgatroyd, L., and Flatley, J. (2001). Harmonised Concepts and Questions for Government Social Surveys, ONS, revised ed., www.statistics.gov/harmony/default.asp.

Manners, T. (1998). Using Blaise in a Survey Organisation Where the Researchers Write the Blaise Datamodels. *Essays on Blaise (1998)*, 5th International Blaise Users Meeting. Lillehammer: Statistics Norway.

Manners, T., Cheesbrough, S., and Diamond, A. (1993.) Integrated field and office editing in Blaise. *Essays on Blaise 1993*, 2nd International Blaise Users Meeting, OPCS (now ONS), London 1993.

Manners, T. (1992). New developments in computer assisted survey methodology for the British Labour Force Survey and other OPCS surveys. *Proceedings, Annual Research Conference*, U.S. Bureau of the Census, 1992.

Pierzchala, M. and Farrant, G. (2000). Helping non-Blaise Programmers to Specify a Blaise Instrument. *Proceedings of the 6th International Blaise Users Conference 2000*, Kinsale, Ireland. Central Statistical Office, Cork.

Pierzchala, M., and Manners, T. (1998). Strategies for Producing CAI instruments for a Program of Related Surveys. In *Computer-assisted Survey Information Collection Methods*. New York, NY: John Wiley and Sons. Presented at the InterCASIC Conference (1996), San Antonio, TX.

Pierzchala, M. (1997). Optimal Screen Design in Blaise. *Essays on Blaise 1997*, 4th International Blaise Users Meeting. Paris: INSEE.

Pierzchala, M. (1996). CAI and Interactive Editing in One System for a Survey in a Multimode Environment. *Proceedings of the Data Editing Workshop and Exposition*. Statistical Policy Working Paper 25, Federal Committee on Statistical Methodology, Statistical Policy Office, Office of Information and Regulatory Affairs, Office of Management and Budget, Washington, DC.

Pierzchala, M. (1995). The 1995 June Area Frame Experience. *Essays on Blaise 1995*, 3rd International Blaise Users Conference. Helsinki: Statistics Finland.

Pierzchala, M. (1992). Generating Multiple Versions of Questionnaires. *Essays on Blaise 1992*, 1st International Blaise Users Meeting. Netherlands: Central Bureau of Statistics.

Schou, R. (1995). Developing a Multi-Mode Survey Processing System. *Essays on Blaise 1995*, 3rd International Blaise Users Meeting. Helsinki: Statistics Finland.

Singleton, N. (2001) Psychiatric Morbidity among Adults 2000 - First Release at www.statistics.gov.uk/pdfrdir/psymorb0701.pdf.

Stanford, V., Altwater, D., and Ziesing, C. (2001). Programming Techniques for Complex Surveys in Blaise. *Proceedings of the 7th International Blaise Users Conference 2001*, Washington, D.C.: Westat.

AFTER 10 YEARS OF CAPI, INSEE EMBARKS ON THE CAPI 3 PROJECT

Christophe Alviset, INSEE

Mario Avi, INSEE

Philippe Meunier, INSEE

Plan:

I – Broad outlines of the CAPI 3 project.

II – The future data collection work-station, a tool for communication

III – Data coding integrated into the data collection work-station ?

I – BROAD OUTLINES OF THE CAPI 3 PROJECT

Following initial tests conducted between 1987 and 1989, it was in 1990 that INSEE (France's National Institute of Statistics and Economic Studies) began work on the CAPI project, with the employment survey. Blaise 2 software was used. The next few years, through to 1999, saw all household surveys gradually transfer over to CAPI. At the end of the last millennium, only surveys for which an electronic questionnaire was not suitable (surveys involving the homeless, regional surveys performed on small samples, etc.) still remained in a paper-questionnaire format.

10 years after, the new CAPI 3 project that INSEE is going to develop, will not create any upheaval in the process of designing and conducting a survey in CAPI. Nevertheless, it is still an ambitious project in terms of the computer-related workload it will generate, as well as the organisational changes involved for statisticians, interviewers and survey managers.

As a result, analysis is already at an advanced stage in terms of the objectives to be achieved, highlighting four aspects of development.

The first aspect will involve providing statisticians and interviewers with an Extranet information system, including an e-mail program, access to the INSEE website and a common forum. The use of electronic cartography and GPS satellite positioning will be tested.

Secondly, the system will incorporate Blaise innovations (Blaise 4 and Windows). Finalisation of the TADEQ project (electronic questionnaire documentation) is eagerly awaited. Survey taking via the Internet will be tested and implemented, together with all the necessary precautions, where relevant.

Thirdly, it will involve introducing developments corresponding to the requirements identified in recent years. We may mention here:

- the possibility, on a statistician-designer work-station, of monitoring data collection at national level;
- organising auditing arrangements, differentiated according to surveys;
- the involvement of statisticians in Blaise programming;
- preparation of a single source data model, for the three different types of use involving the interviewer, the survey manager at regional level and the statistician-designer;
- integration into CAPI, of the management of the samples by the interviewer and supply of the data required for the purpose of their remuneration;
- construction of a standard survey kit for regional surveys.

Lastly, the final theme relates to developments that might be implemented following validation by an in-depth organisational study. This will involve the deployment of CATI, which has not so far been used at INSEE, or the routine introduction of coding at the source, i.e. on the data collection work-station. The idea of coding at the source, which has been much debated among statisticians, has been the subject of various consultations with designers, a summary of which will be found in Part Three of this paper.

II – THE FUTURE DATA COLLECTION WORK-STATION, A TOOL FOR COMMUNICATION

While the vast majority of INSEE interviewers have liked the new method of electronic data collection, using Blaise software, over the course of time, they have not failed to denounce its imperfections and weaknesses, both individually and collectively. This led INSEE, from 1998 onwards, to initiate various studies (audits) aimed at producing an assessment of the transfer of surveys over to CAPI. The interviewers' reaction was a unanimous one: the idea of returning to the paper questionnaire was simply out of the question! This assessment led to the groundwork being laid for a new project, CAPI 3, which is both intended to make up for the weaknesses of the current system and to innovate strongly, by incorporating everything that appears to be sufficiently tried, tested and reliable among the new technologies.

The starting point for this analysis was to allow any INSEE employee and by extension, any interviewer employed on temporary contracts to access the INSEE information system from outside, to enable them to do their job better. This implies both an analysis of the type of work-station and of communication requirements, while bearing in mind constraints governing authorisation for access and security.

If we are to achieve this goal, we shall need a clear vision of the software and hardware systems to be deployed, following an exploratory study of the new information and communication technologies, in order to arrive at a documented statement of requirements.

To prevent us being overtaken by day-to-day events and constraints, it was decided to organise 2 round table sessions, with ten or so interviewers being invited to each session, in order to catalogue their experience, their requirements and their expectations, and to compare them with the Institute's own objectives and the currently available technological possibilities.

The main expectations expressed by CAPI interviewers can be listed under three headings:

- a very strong need for communication with INSEE and with fellow colleagues;
- simplifying the tasks of pinpointing households on the ground and gaining access;
- easy-to-use, tried and tested technologies.

II - 1 – A VERY STRONG NEED FOR COMMUNICATION WITH INSEE AND WITH FELLOW COLLEAGUES

This is not a new requirement. Internal communication at INSEE, as within other public-sector organisations and the private sector, involves the widespread use of Intranet systems. Thanks to the extranet, these systems are gradually being extended to allow their use by external partners. They could be opened up to freelance interviewers. This would allow the installation of an open e-mail program functioning with INSEE and other interviewer colleagues, globalisable and programmable information transfers, and forums.

At the moment, interviewers have an e-mail program integrated into the CAPI application on their work-station,. However, this is a very restrictive and cumbersome method of communication: it only allows each interviewer to communicate with the establishment of the region they work in, and moreover, within the framework of a single survey (there are as many e-mail programs as there are surveys, being managed simultaneously on a single work-station).

The new project involves providing interviewers with a universal e-mail program, similar to Microsoft Outlook, enabling them to communicate both with one another and with their managers, independently of any work in progress and allowing them to log on to the central systems (e-mail program, databases, applications) in real time, from wherever they happen to be at a particular time.

Communication between interviewers, in addition to the social aspect, should contribute to the exchange of information, assistance and advice, and thus consolidate the feeling of being involved in a team effort. Interviewers should be able to access an indicator showing the overall amount of progress made by the survey in which they are taking part, so that they can judge their own progress. Having a direct link with regional management assumes that the latter will be available and able to react promptly. Management should certainly be capable of meeting an urgent need, and providing logistical support by remedying any technical problems within 24 hours, thanks to contracts signed with suppliers. No doubt this calls for harmonisation of regional staff on-call policies, or maybe even for support to be provided at national level, rather than having local staff on call. This is the appropriate response to the very strong need for communication with INSEE and with fellow colleagues. It has to help remove the feeling of isolation encountered in situations involving difficult surveys (e.g. sociological surveys such as disability-invalidity-dependence). The widespread provision of mobile phones to interviewers, which should go hand-in-hand with comprehensive geographical cover in terms of accessibility, will round off these arrangements.

This communication flow will also help to strengthen the bonds of trust with INSEE. It will carry offers of work from regional management and transmit urgent information. The data disseminated by interviewers will include expenses claims, any supporting documentation requested and feedback involving follow-up and difficulties encountered. Among interviewers, it will allow information and feedback to be shared quickly, and provide help and assistance for new interviewers in the field (guidance and mentoring), as well as accompanying them in the field for a day, in order to understand the nature of the area concerned. It will be a way to provide release from stress and tension, by sharing experiences.

These arrangements will go hand-in-hand with the organisation of periodic meetings, systematic survey review meetings, annual meetings, sharing of information and exchanging experience among small groups (useful tips and advice), internal cohesion seminars, and pleasant moments shared with fellow colleagues. It will be supplemented by the provision of different media for interviewers: INSEE documentation, brochures and leaflets.

The same applies to communication in general. Nevertheless, interviewers have to download their own programs covering surveys and special procedures, such as anti-virus software updates, and transmit their completed electronic questionnaires. This involves the globalisable and programmable transfer of information. At present, this transfer takes place survey by survey, from the interviewer's

own home, via their telephone line, manually (using a non-programmable method). Hence the complaints made by interviewers, who see this as a constraint and are deprived of the use of their own telephone line during the transfer period, even though the cost of the call itself is, of course, paid for by INSEE. It would therefore be desirable initially to be able to program these transmissions so that they can take place at night for example, and to globalise them, e.g. by automatically interlinking any transfers of completed questionnaires related to different surveys and program downloads. At the same time, it may be possible to develop wireless (WAP-type) transmissions sent from mobile phones, as soon as the performance of this technology reaches an acceptable level.

A forum would involve both creating a place for exchanges and discussion between colleagues (where they can exchange useful tips and advice, for example) and providing all the data that interviewers may need (programs, procedures, updates). To some extent, it is therefore a variation of the systems described previously.

Finally, within the framework of the survey system and the intermittent working pattern followed by most interviewers, via appropriate identification we need to be able to limit the access rights of every interviewer to the periods defined by the contract of employment binding them to INSEE.

II - 2 – SIMPLIFYING THE TASKS OF PINPOINTING HOUSEHOLDS IN THE FIELD AND GAINING ACCESS

Paradoxically, the problems linked to pinpointing the dwellings surveyed did not give rise to any particular comments in terms of the pinpointing techniques used. These were not formally requested and a number of new difficulties will undoubtedly emerge with the implementation over the next few years of a new method of conducting an ongoing census of the population. Part of this census will be conducted exhaustively, and part by conducting a survey (municipalities of 10,000 inhabitants or more).

Pinpointing of the dwellings to be surveyed relies on two sources and is based on two sampling techniques. The two sources are the survey frame constituted by the last census performed and the administrative file of new dwellings. Both techniques consist of a sampling involving several degrees known as the master sample (applied to most households surveys other than employment surveys) and an areolar sampling made up of areas containing about 20 dwellings (in the case of employment surveys). The problems arising differ in terms of their nature and their scale in both cases. The first case involves pinpointing a specific dwelling, while the other case involves accurately circumscribing the contours of a geographical area containing about twenty dwellings and locating them all within this area.

Until quite recently, regional management was in possession of all the paper documents related to the last census (1990). These documents enabled the search for a dwelling that was being surveyed but was not clearly identified, to be refined, as needed, via the consultation of adjacent documents. From now on, this will no longer be possible. Paper documents are being centralised and transcribed onto magnetic media using optical character-reading techniques. The information required for pinpointing dwellings to be surveyed (address cards) is printed out centrally and sent to regional management. A certain loss of clarity may therefore be associated with this transcription (which we are endeavouring to reduce as far as possible) and in the event of a difficulty, it will not be possible to search the surrounding area.

Assistance with pinpointing using the GPS (global positioning system) or any equivalent system, associated with electronic cartography, would no doubt merit consideration. This has not been the case in the past, since no genuine requirement has ever been ascertained and maybe also because the pinpointing involved was not regarded as sufficiently accurate. Indeed, GPS is probably not accurate enough for urban use. A test conducted in the department of La Réunion for the purpose of the census was inconclusive and the idea was abandoned. However, this department uses geographic information system software, loaded onto a laptop.

Finally, in the future, and as has just been stated, the censuses conducted in France will employ a new methodology (the updated population census or UPC), functioning continuously, and involving a technique of surveying 40% of the dwellings in an urban environment (towns with over 10,000 inhabitants). GPS-type pinpointing would also be able to offer assistance here, that would merit analysis, provided sufficient accuracy can be achieved.

On the other hand, interviewers complain quite rightly about the inaccuracy of the pinpointing information provided to them and are requesting "up-to-date" address cards. They would like to have a software program that can search for addresses without any other details, lists of residents living in a particular block of flats (a study to be performed in connection with the future updated population census), information that will help them optimise their rounds of household visits, provide street maps and passes (of the type used by post-office employees when they distribute the mail) allowing them access to buildings with door entry codes. They would like an electronic phonebook function on their work-station, and street maps of large municipalities, initially on their laptops, and later on their personal assistants (see below).

In order to facilitate access to the household being surveyed, interviewers will be given mobile phones for the purpose of making contact, particularly where they find themselves on the other side of a closed door.

They are also requesting a system to help them optimise their rounds of household visits: a study is envisaged with a view to examining a solution based on a personal assistant, if this can incorporate address cards and visit record books.

The mobile phone could be used to number the series of pre-recorded calls made from the laptop (e.g. for those types of surveys, such as the ongoing employment survey, which involve an initial face-to-face interview, followed by several telephone interviews, one after the other).

A notification letter is systematically sent out, as part of the arrangements for making contact with the household surveyed. The Post office is putting in place a new system aimed at offering traceability for mail - "the tracked letter". This system relies on each postman entering the barcode of the letter he is distributing, and it should be operational in 2003. Hence the possibility of accessing this information i.e. tracking the progress of any letter posted. It will be tested in order to assess its benefits and effectiveness.

II - 3 – EASY-TO-USE, TRIED AND TESTED TECHNOLOGIES

Among all the difficulties reported by interviewers in carrying out their work there is one that is linked primarily to the hardware used, i.e. the laptop: it weighs too much, the batteries don't last very long before they need recharging, it is not always reliable, it takes an excessive amount of time to power up and load questionnaires, it is difficult to change from one survey to another, and data transmissions take a long time and are subject to frequent incidents. However, as mentioned above, no interviewers want to return to the paper questionnaire.

Advances in the Blaise software and the advent of Windows 2000 have considerably improved the questioning process. The progress made over the past 10 years has also improved the performance of laptops, and allowed data transmissions to be optimised and become more reliable. However, weight is still a problem, because in addition to the laptop itself, there is the mass of paper documents that interviewers have to carry around with them - address cards, documentation linked to the survey, etc.

New personal assistant-type tools, which will be also tested within the framework of preparations for the new updated population census, might be able to incorporate address cards and visit record books into personal assistants and thus considerably reduce the weight of paperwork that interviewers have to carry around.

It will be advantageous to study their convergence with mobile telephony, in order to simplify the hardware provided to interviewers.

Later on, a new generation of ultra-light laptops with a removable flash memory will no doubt become accessible, and its benefits should include enhanced data security, since the flash card can be removed from the computer after use.

Advances in investigation methods linked to an expansion of the scope of issues dealt with by INSEE to encompass social affairs, will require new functions to be available on laptops. One such example is data acquisition via an optical barcode reader, which is already possible with Blaise 4 and is undergoing testing at INSEE with the future Health survey in mind. However, the main function involved here is audio recording and playback, for some types of interviews, to allow high-quality listening (playing various sentences or texts to people and asking them to respond with their own interpretation). While observing safety and reliability requirements, it will also be necessary to gain partial access to the inside of the laptop for the purpose of carrying out exercises, supporting other applications, playing self-tuition CDs, etc.

While considerably reducing the burden of paperwork to be carried around and handled, the multiplicity of functions that can be supported by laptops and their associated hardware reinforces the need for interviewers to be able to print out from their work-station.

Finally, some interviewers are advocating the benefit of a webcam to facilitate the sharing of information with colleagues and/or their regional management (though not in connection with surveys).

It is interesting to note that interviewers make the point that the technologies implemented must be tried and tested, simple, easy to use, flexible and open-ended, so that they can accommodate progress. They must be accompanied by appropriate training, with logistics and support facilities consistent with the development of the technological resources themselves (timely repairs in the event of a technical problem). For the purpose of diagnostic analysis of any incidents, we should have a black box that could retrieve all the manipulations that have taken place.

III – DATA CODING INCLUDED ON THE DATA COLLECTION WORK-STATION ?

This involves implementing a standardised coding system on the data collection work-station, which will either be assisted or automatic, for textual answers provided by survey subjects during the course of the interview. This is therefore a system of coding "at the source". The aim is to cut costs, by removing the coding stage, which at present is carried out after data collection, and to cut lead-times by providing the survey designer with the results as soon as the survey is completed.

In fact coding is already in use on the interviewer's work-station in the case of certain questions that involve highly standardised answers (type of road, code of the municipality, department, country of birth, etc.). However, the CAPI 3 project has raised the question of a standardised coding system for all textual answers, which amounts to entrusting the entire task of coding to the interviewer, using the coding software on his or her data collection work-station.

It will be seen that after considering the benefits and disadvantages of this system with survey designers, it does not appear to be necessary for full coding at the source to be routinely included on interviewers' work-stations. Its implementation can only be partial, based on the degree of complexity of each response contained in the survey.

III - 1 – ARE PRODUCTIVITY BENEFITS REALLY ACHIEVED OR IS THE WORKLOAD MERELY TRANSFERRED UPSTREAM ?

The routine introduction of coding at the source would abolish the tasks of monitoring and coding, which are performed after interviewers have dispatched their survey data to INSEE's regional management offices. However, the workload of the interviewer during the course of the interview, on the other hand, would be increased.

First of all, before conducting the survey, he will need to be trained more thoroughly than is the case today in the various classifications. These may be complex, and their hierarchical structure and detailed contents have to be understood. Admittedly, fully automatic coding would allow all answers to be processed, though at the price of rejects that would have to be recycled, as they are at present, downstream from data collection.

The interview will also take longer, leading to a higher cost of data collection. Either coding will simply be assisted, and the interviewer will have to finalise the choices on offer, or it will be fully automated, with software that will necessarily have to be powerful plus a vast reference system, taking up machine-time and increasing the length of idle periods during the interview.

Moreover, any productivity benefits due to the expertise of coding specialists at regional-management level would be lost. Conversely, there would be an improvement in quality because the interviewer could immediately specify the household's answers in cases where several options are on offer – provided that he does not influence the answers given by the survey subject too strongly.

III - 2 – SHORTER LEAD-TIMES FOR PRODUCING THE RESULTS, BUT TO THE DETRIMENT OF QUALITY ?

The second objective of standardised coding at the source, i.e. on the data collection work-station, would be a reduction in the lead-times required to provide the survey designer with the data collected. This objective should indeed be achieved - except in cases that involve the processing of automatic coding rejects, which is carried out downstream from data collection. Whether or not the data is checked subsequently to data collection, the centralised coding phase following the survey would be considerably reduced.

Consequently, there is a reduction in lead-times, but undoubtedly to the detriment of quality.

The main objection, from a statistical point of view, to coding at the source is distortion of the truthfulness of answers by interviewers, via their own prism through which they apprehend reality. They would appear to direct the answer to match their own vision and knowledge. For example, as regards a household's occupational activity, they would tend to distort this by coding it to match their own social category, according to the jobs they are familiar with. With assisted coding, they would tend to force this issue, thus introducing a bias and wasting time. This tendency might be reinforced by the survey subject's own hesitations, as he does not always know, for example, which sector of activity his employer is involved in (it would be possible to introduce a non-blocking control onto the work-station, activating a directory of all establishments and their business code, but this would burden the machine-time considerably). Whether it is automatic or assisted, coding by interviewers harbours the danger that they will truncate the answers given by survey subjects. Interviewers tend to standardise the answers given, which in turn tends to impoverish the variety of different situations. If new jobs and new diplomas come into being, interviewers will tend to code them within the current nomenclature. Thus, the task of determining the occupation of the survey subject would seem to correspond to their own knowledge of socio-professional categories, without being able to apprehend either new jobs or distinctions calling for a technical knowledge of the sector of activity in which the survey subject is involved. What tends to happen is a "smoothing-out" of the answers, which distorts the reality. Nor shall we discuss here special questions, e.g. those concerning diseases, the answers to which can only be coded by health experts.

The result of "smoothing out" answers would seem to be the absence of a gradual enrichment of the learning base, due to the interviewer's lack of expertise in these areas. For example, as far as the classification of socio-professional categories is concerned, we would see changes in occupations escaping our grasp, and we would no longer identify new occupations or defects in our own classifications. At present, we are safe as regards this situation, since the departments responsible for coding enrich the database with any changes they detect in the answers provided by survey subjects. However, one of the leading objectives of many surveys is precisely to determine transformations taking place in society, in its activities, behaviour and opinions.

Without using standardised coding at the source, one very useful tool on the data collection work-station - and fairly inexpensive to install - would be a spell-checker, which would not include any blocking control. This would be used mainly for the purpose of clearly entering the occupation of the survey subject and would reduce the failures in the automatic coding process currently applied downstream from data collection. It would also be used for clearly entering the nature of the business of the employer of the survey subject (or a household member). Lastly, it could be tested on the surname and first name of the address, but the database would require a substantial capacity.

Another objection to coding at the source is that the time involved in coding increases the length of the interview, either in order to obtain detailed information - related to the progress of the software - or as a result of idle periods while the software is functioning. Clearly, this is harmful to quality. Interview length is a key factor in any survey, the aim being to limit this to a reasonable time. Increasing the length of the interview is likely to tax the patience of the household before the interview is over, idle periods are not used by the interviewer for discussions with the household,

causing him to lose concentration, and the information required by the software might be other more important questions for the purpose of the survey.

III - 3 – CONCLUSION: THE INTERVIEWER MUST CONCENTRATE ON HIS KEY TASK: CONDUCTING AN INTERVIEW LEADING TO THE RIGHT ANSWERS

Full coding at the source, which is thought to generate productivity benefits and shorter lead-times for making the data collected available, adds an extra task onto the interviewer's key function, which is to obtain satisfactory answers, i.e. answers that do not distort reality. There would be idle periods during the coding operation, during which the interviewer would not be engaged in talking to the household. This situation is unacceptable: any time spent with the household must be devoted exclusively to them, as the interviewer's task is to conduct a dialogue with the household. The interviewer is too tense to gather information during the interview because he is concentrating on other things apart from conducting the interview. Automatic coding – which is a cumbersome task - can therefore only be envisaged away from the household.

If benefits are to be gained, it is on this focal point, reducing the survey time while maintaining the quality of data collection. Coding is also somewhat removed from the interviewer's own job. A good interviewer may not necessarily have an aptitude for coding. This would involve combining two jobs, and certainly to the detriment of overall performance.

Having said that, a certain amount of coding already exists on the work-stations of INSEE interviewers. It is straightforward, and well accepted. It is based on straightforward, known classifications. Empirical rules are suggested by survey designers:

- During the course of the interview, do not introduce coding exercises that take up a large amount of time (e.g. socio-professional category). No variable containing over 60 headings should be coded, otherwise, the coding operation would take too long, introducing an idle period into the interview. From a practical point of view, coding must take no more than 15 seconds; beyond this time, it breaks up the interview.
- Types of nomenclatures that the interviewer cannot itemise:
- Too many headings (see supra), involving too much complexity (numerous root structures) and thus time;
- Phenomena that the interviewer is unable to code (e.g. diseases, pharmaceuticals).

Seeking to make matters any more complex when visiting the household would distort the meaning of data collection. In fact, the problem is similar to that of the controls located on the data collection work-station: they are effective if they are limited in terms of their number and complexity.

Challenges of Instrument Development

William E. Dyer, Jr., U.S. Census Bureau

Ellen Soper, U.S. Census Bureau

**Bureau of the Census
4700 Silver Hill Rd Room 3640
Suitland, Maryland 20746**

The Census Bureau's Technologies Management Office (TMO) Authoring Staff consists of a centralized team of programmers (a.k.a. authors) who develop Computer Assisted Interviewing (CAI) instruments using two different authoring languages, i.e., Blaise for Windows and CASES written for DOS. The Authoring Staff has written CASES instruments for years, but has only recently been developing instruments for Blaise. As more and more instruments are written for Blaise, new tools and standardized approaches are desired to help streamline project management, instrument development, testing and deployment.

The Bureau of Census (BOC) is a large organization that historically fields large complex surveys. The automation of large complex surveys introduces additional challenges to the process of data collection. The purpose of this paper is to identify some of the aspects of authoring that we perceive as being challenging and to review the approaches we have taken to deal with these issues. This paper will focus specifically on the challenges associated with project management, instrument development, instrument testing, and training of personnel.

PROJECT MANAGEMENT

Challenges

There are several challenges associated with managing a survey automation project. From a software development point of view, it is important to recognize the need to improve proper project management procedures to ensure all the tasks associated with programming culminate in a product that meets requirements within the established time frame and within budget. The project manager's role of getting a good understanding of the software requirements is a difficult challenge. More challenging is dealing with requirements that continually change throughout the lifecycle of a project. Effectively controlling change so that the project remains on schedule is an important issue and must be dealt with. The ability to establish and maintain good customer relations is also an important issue that should not be overlooked. After all, customer satisfaction is a key factor to success. How does one manage a project to a successful conclusion while at the same time ensuring the software developers receive what they need to perform technically, within schedule, and in concert with one another?

Approaches

How do we get good requirements?

A typical survey automation project will have internal (BOC) subject matter specialists who serve as the overall program manager, otherwise known as the sponsor. One of the sponsor's roles is to gather requirements from the client, such as Bureau of Labor Statistics or Bureau of Justice Statistics or another outside agency. The sponsor will then apply their survey automation experience to develop program or instrument specifications. The program developers work from these specifications to develop code that meets the requirements. It is important that the programmers and sponsors work very closely to identify

the requirements and any ambiguities in the specifications. It is critical to establish and maintain a common understanding of the client's requirements with the client and all other project members.

We have learned that specifications that were written with one software language in mind, e.g. CASES, should not be used as specifications for another software language, e.g., Blaise 4 Windows. The software language architectures are different which could result in limiting the instrument functionality. For example, designing tables that have arrays of arrays may be conceptually correct but impractical to implement. Some would argue that the ideal specification is developed such that it is not dependent on any software language. However, we have found that the best specification is one that is developed with programmer involvement that includes structures similar to the software language that will be used for development. We recommend the specification include information about the desired block structure, edit conditions such as checks and signals, and detailed logic statements within the rules sections, to name a few Blaise specific software attributes. A software language specific specification helps with the programming effort and improves the software development process.

How do we manage a project for success while keeping the customer satisfied?

Once the requirements are clear, it is important to create and disseminate a software development plan. The software development plan serves to educate the sponsors on the issues related to survey automation. In our experience, the more knowledgeable the sponsors are with regard to survey automation issues the better the project runs. The software development plan serves as a planning and communication tool that can be used to document and manage changes to the plan. A well-documented plan defines expectations for all project members and helps to establish the timeline. The plan should also include how communication will be handled. It is important to identify roles and responsibilities and determine who will have authority over what aspect of the project. When clear goals are established and communicated there is a probability the objectives will be achieved on time and within budget.

How do we deal with change control?

It is often the case that more than one person is assigned the task of developing instrument code. The coordination of programming activities can be a challenge without proper configuration management techniques. At a minimum, requirements, specifications, and program code need to be managed in order to maintain their integrity and traceability. The sponsor and the Authoring staff share in this responsibility. Several survey sponsors have developed specification databases to store metadata about the survey questions that will be used for data collection. The sponsor will populate the database with information that is useful for their own purposes such as edits used in post processing of the data as well as the attributes needed to provide an instrument specification. Changes to the instrument specification are maintained in this database and new specifications can be output at the press of a button.

Additionally, change requests are stored in a centralized database system that both sponsors and authors access. This system is referred as the Change Request Tracking System. This system is used once the instrument enters into the testing phase of software development. Changes that are identified as a result of testing whether it is a new requirement or a problem identified in the instrument and are entered into the Change Request Tracking System. The time, version, nature of the problem and priority are all assigned. The software developers access the system to review the changes and to update the resolution of the change. The sponsors access the system to review the status of the change, enter new changes, or to update the existing requests. This facilitates making changes efficiently and ensures accountability on both sides. The change control processes should include a feedback loop when a request cannot be accommodated, is not clearly understood or cannot be addressed within the requested time period. Under these circumstances an author can make notes or provide explanations directly into the Change Request Tracking System. The change request database stores requests for modifications to the instrument that can be reviewed by the project manager and distributed for action. Time and date stamps are used to verify that the sponsors needs and requests are being attended to and that each project member is aware of

their responsibility. Deadlines for submitting changes can be documented, incorporated into the project plan, scheduled and measured. The Change Request Tracking System has allowed us to do research into the number and types of changes being requested and applied so we can identify opportunities for improvement. Having a way to document requests, prioritize and disseminate them to authors turned out to be a real time saver. This system also serves as an excellent reference for future or past projects.

Another method of change control is the use of software to archive different versions of the source code. Program files are stored in a version control application. Through proper use of the file check out/check in concept, program changes are captured, maintained, and problems with sharing program code are minimized. These files as well as shared source code can be placed under version control and locked by the person who has responsibility for insuring their integrity. Establishing ownership of files is necessary when multiple programmers are working on a project.

DEVELOPMENT

Challenges

A challenge in developing instruments for multiple surveys is ensuring programming efforts are coordinated and applied consistently across the board. It is difficult to foster a common understanding of the development tasks that occur across surveys for all team members. It is easy to develop the same functionality different ways when you have different programmers and specification writers working on the same and/or different projects. By having a common understanding of how requirements have been met, we can apply past practices to achieve continuity across instruments. Managing program files is also a challenge that affects a team of software developers who work on multiple concurrent projects. It is important to develop and organize a system that will work for all projects and programmers. The Census Bureau develops surveys, which are heavily customized to meet sponsor requirements. Sometimes we would like to modify the software default behavior requiring manual intervention by a programmer. Instrument preparation and release becomes a special challenge with the large volume of files that are required. It is essential that instrument builds are performed correctly, are consistent between builds and are completed according to established schedules.

Approaches

How is a common understanding of development tasks achieved?

We have taken several approaches that attempt to provide a universal understanding of development tasks. The Authoring Staff has approached the development of multiple Blaise instruments by organizing a team of programmers who will initially design and develop the instrument. At some point, the code is handed off to another programmer to maintain or enhance. This arrangement allows the same programmers to develop the more complex programming tasks consistently from one project to another providing continuity across surveys. The project manager is part of this team of initial developers. The project manager manages activities associated with developing complex instruments by working with software developers to create standards, libraries and procedures that affect similar tasks across one or more projects. It becomes easier to identify potential areas to standardize when the same resources are performing the tasks.

Incorporating new Blaise programming standards and procedures into our existing infrastructure improves future instrument development efforts. All programmers share information with one another in regular weekly meetings organized to provide updates on the project status. These meetings are an opportunity for software developers to share techniques or to discuss issues that relate to meeting the software requirements. Many opportunities have arisen that have led to development of code that can be reused. For example, we have developed external databases that hold fill data but are called using a

simple procedure. The same procedure is used throughout the instrument minimizing coding. Using lookup tables rather than hard coding fills allows us to change the external file without having to prepare the instrument.

To facilitate handing code from the initial developer to another programmer for maintenance it becomes important to create libraries of shareable code. A centralized library allows groups of programmers to coordinate activities and further simplifies the development process. We have found that a project can share many files without conflict as long as a single person with an assigned back up contact maintains the library. One of our more successful techniques to share information was to publish a “tips and pointers” document in the centralized library. This library also includes documents such as programming standards that include generic naming conventions such as those used to define Blaise types and block structures. This simple method for standardizing type and block names helps the programming staff to develop modules of code that could be worked by any member of the staff.

How are files managed?

Blaise 4 Windows offers many new features not found in DOS-based systems. As a result, it is often the case that there will be more files that need to be managed. Standardization of the development environment, i.e., directory structure, is one way to manage files and ultimately, improve the process of developing instruments. We have consulted with both Westat and Statistics Canada about how they have implemented the directory structure for Blaise instrument development and field operations. In combination with recommendations from several areas, we have a “recommended” directory structure that can be used for our instrument development from cradle to grave. It seems that there is no one perfect way to organize files. If the single directory approach is taken, large instruments become unmanageable. If a tiered directory structure is used, the program files are better organized but version control issues are more pronounced. Since some files are used strictly for development and others are used at runtime, we have attempted to organize somewhat by the nature of the file type being used. Source code modules, libraries, type definitions and procedures are subdivided into functional directories. In addition, files for bitmaps and lookup tables are put into a directory called ‘externals’ that can be set as read only at the directory level to enjoy better performance and simplify code management. We have split our efforts into a development area and a build area to keep from conflicting with each other’s work and further layered the build to support multiple versions of the release over time.

How are instruments prepared and released?

Building instruments in Blaise, even with the Blaise control center software tool, is a complex process with a lot of pieces that need to be arranged to make all the parts work together as an integrated instrument. There are advantages in having build and release procedures to control the process of preparing instruments. The key to success when preparing an instrument for release is to remember all the steps, the order that they need to be performed and all the details involved with setting the ModeLib, key bindings, help files, Manipula programs, data model properties, etc. It is unrealistic to expect one person to be solely responsible for this process, so we captured the steps required and trained others in the ways of building instruments in Blaise. This was a major departure from what authors have been expected to do for other languages. To this day we are having ongoing discussions about the best ways to implement the build procedure. We have found there are some manual steps required in the preparation process of Blaise instruments that cause unexpected results if not properly performed. Each time an instrument data model is renamed, which is a Census Bureau requirement for each release to production, many configuration items need to be manually reset from within the Blaise Control Center. For example, accept the hidden setting for enumerated fields is a manual step that if not selected will cause enumerated values to scroll off the displayed area. It is our desire to minimize these types of manual changes.

Repeatable results can be achieved by using the same procedures each time the instrument is prepared and released. Since there are so many things to remember when performing an instrument build, we have

made an effort to automate portions of the process. Having the same person do the build every time is impractical so we prefer the team approach of performing tasks. Making check lists helps, but falls short of ensuring the repeatable results we are seeking. Our effort starts with having an experienced programmer, who is familiar with Blaise requirements and the project constraints, mentor other staff in the procedure required to build an integrated instrument. As the steps become well defined we use a combination of written release procedures and automated procedures whenever possible. This benefits us in many ways; we train programmers, provide better documentation and improve the automated release procedures that can be adapted to suit future projects.

As one can see we use a variety of methods and procedures, ways to share code and engineer for code reuse to get the most out of our development efforts. By adopting standards whenever possible for everything from naming conventions to screen design we can have a number of authors working simultaneously on a very large survey. Having both written and automated build procedures allows us to cross train our staff and repeatedly produce quality surveys.

TESTING

Challenges

A centralized programming staff faces the challenge of ensuring instruments are thoroughly tested in an efficient and timely manner. Testing of instruments is a shared responsibility between the programming staff and the sponsors who develop the specifications/requirements. Meeting sponsor expectations of implementing changes quickly is a difficult but necessary task. It is important to develop a strategy of how an instrument will be modified, released and tested. Since we develop multiple instruments with multiple iterations of testing, it becomes necessary to consider the time it takes to make instruments available to the sponsor so they can conduct their testing in a timely fashion. The way the instrument is prepared and packaged, how it is delivered, and on what platform it is run are factors to consider. The TMO Authoring Staff has developed a system that allows sponsors to test the instruments; however, maintaining and supporting this centralized test system for users on numerous platforms, at remote locations, and for multiple software languages introduces additional challenges. For a system that was originally developed to aid software developers in the testing and debugging of instruments, maintaining and supporting this application across many sponsors is a challenge. The primary responsibility of the software developer is to develop the survey instrument. Supporting and maintaining an application that meets everyone's requirements can be time consuming. However, with that said, the centralized test system has reduced the time required to get the instrument to the sponsor or tester.

Approaches

What is our strategy or life cycle model of software development?

Perhaps the most difficult challenge we face is the issue of testing. Often overlooked and misunderstood, testing should be given more time on the schedules, should be better organized during all phases of the project and could become a focal point for improving our processes. Finding and correcting errors early is paramount if an organization wants to have instruments delivered on time and error free.

There are various life cycle models, which integrate well into the paradigm of software development of instruments. Most project managers and programmers are familiar with the classic Waterfall Model, the Iterative Model, and Incremental Model as diagrammed in Whitten, Neal, 1995. Managing Software Development Projects. The typical software development project at the Census Bureau is a combination of these models. Let's briefly review each software development model and discuss how our software development of instruments affects our approach to testing.

Waterfall Model

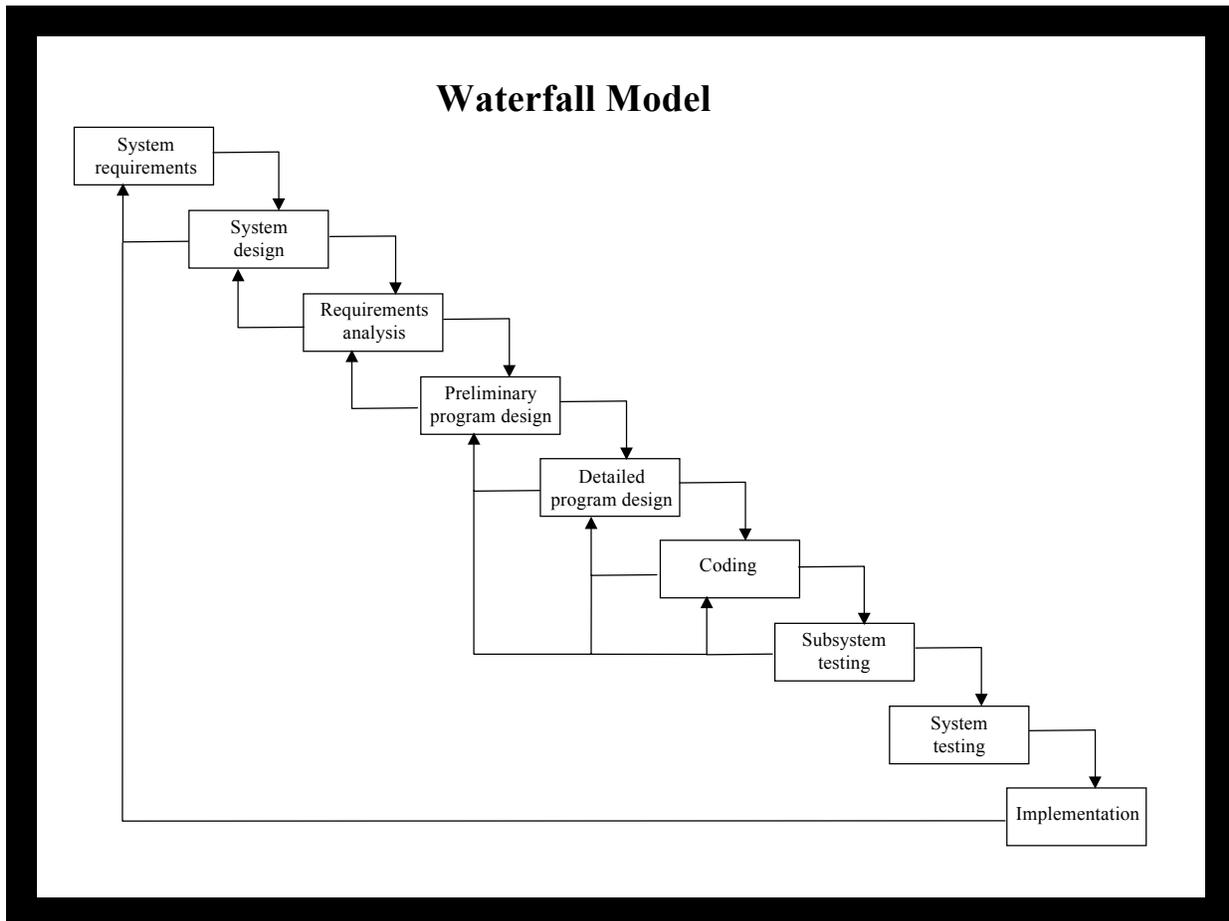


Figure 1

The classic waterfall model of the software development life cycle is very methodical. Typically, requirements are frozen once they are initially agreed upon and changes to the requirements are minimal. Also, testing is done later in the process. This model is especially good for software development when the requirements are well defined and well known up front. See Figure 1.

Iterative Model

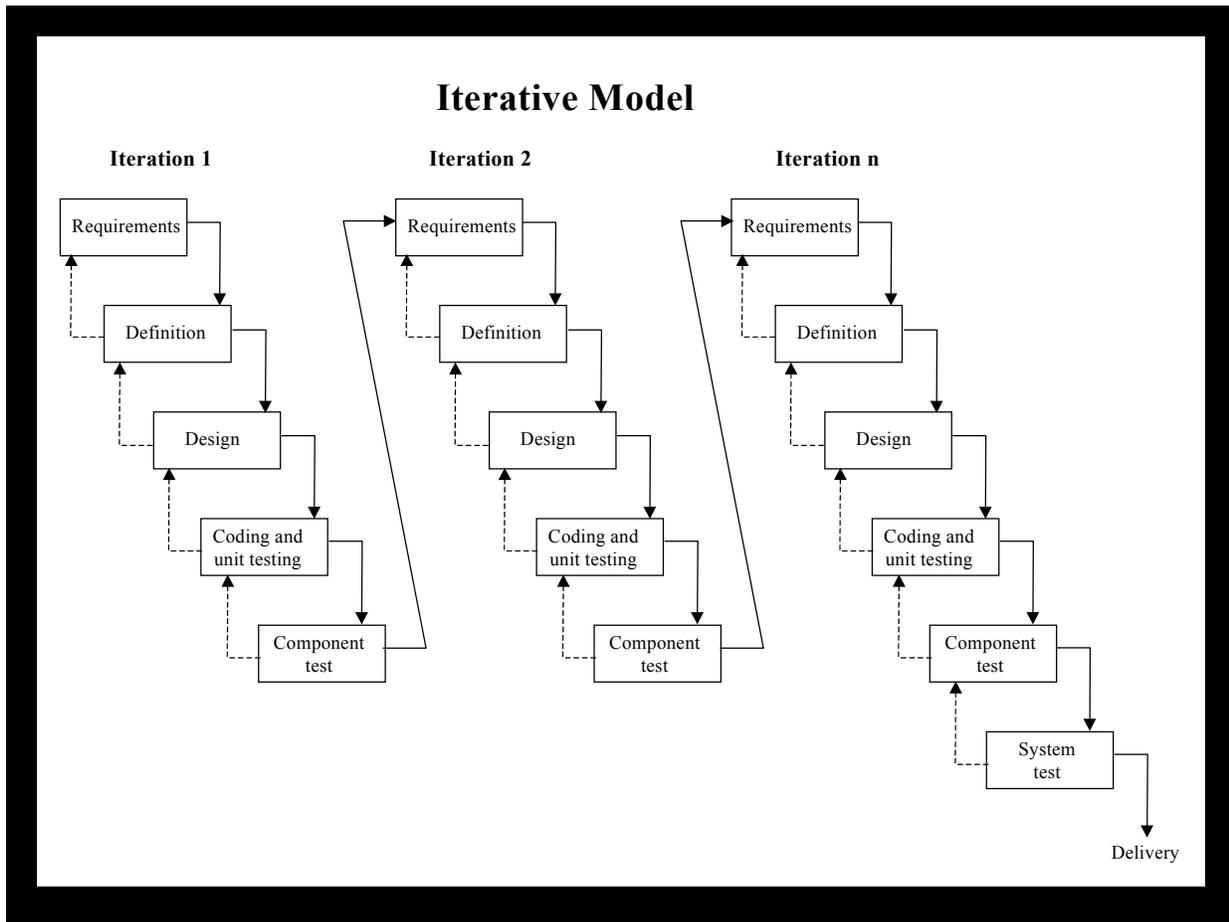


Figure 2

The iterative model of the software development life cycle allows for multiple iterations of the classic waterfall model. Each time an iteration is completed, this model allows for a user test of a baseline product (i.e., a component test on the diagram). The comments that result from the user test are added to the existing requirements to develop the new requirements for the next iteration. This model is especially good for software development when the requirements are not well defined up front because it allows for the discovery of new requirements as you go through the development cycle. See Figure 2.

Historically, this was the software development life cycle used for instrument development. Instrument components were developed in a linear fashion and added as the next software baseline. Components were not often developed concurrent to one another because it was difficult to isolate the code once the components were integrated. We still use this model for most of our DOS based instrument development efforts.

Incremental Model

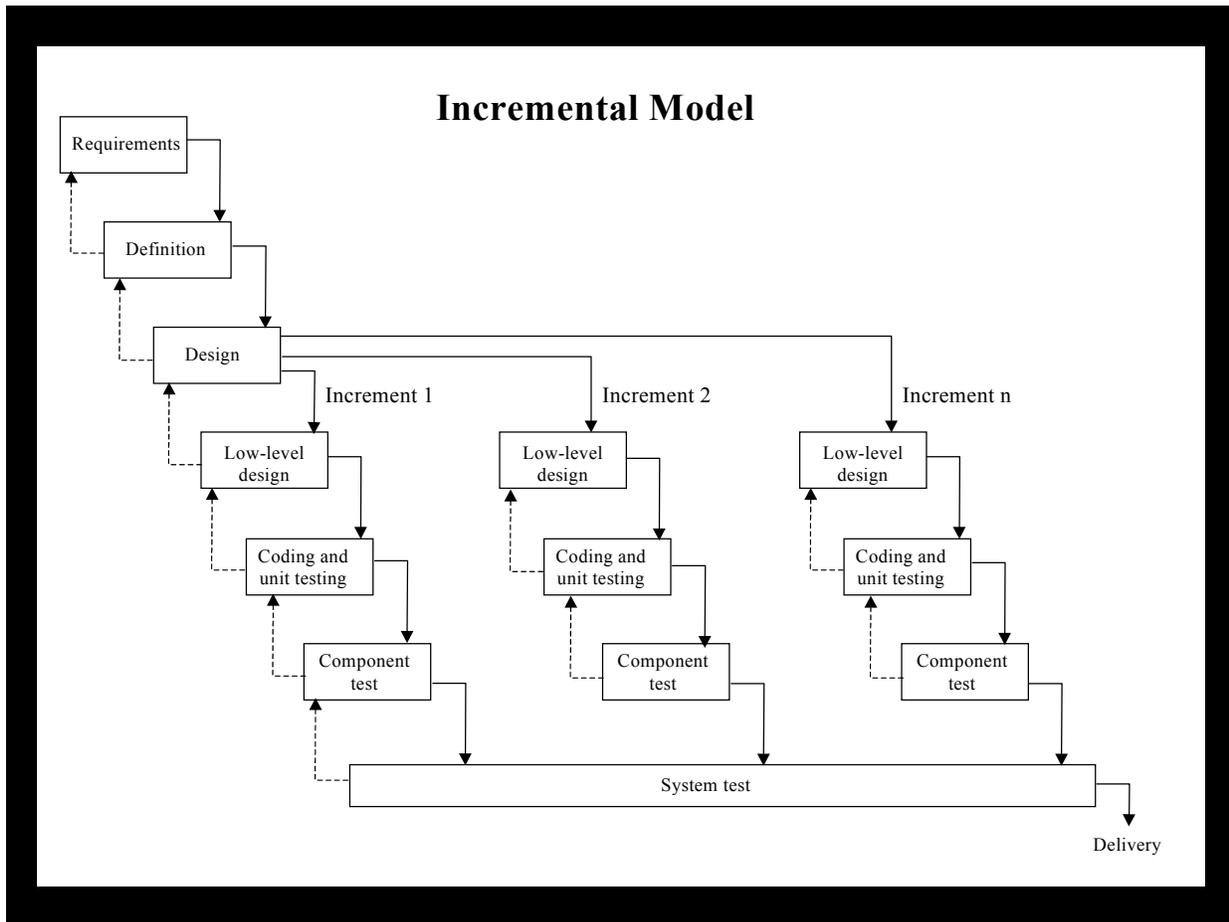


Figure 3

The incremental model of the software development life cycle is essentially a compressed version of the classic waterfall model. Typically, requirements are frozen once they are initially agreed upon and changes to the requirements are minimal. Once an overall design is completed, the development effort is divided into modules or increments so that lower level design, development, and testing can be done concurrently on the modules. The modules are then brought together at the end of the development cycle to be integrated and system tested prior to delivery. This model may be especially good for applications where there are natural divisions for coding purposes (e.g., client server applications). See Figure 3.

As we began developing with the Blaise 4 Windows software, we learned that the software has certain organizational flexibility we could use. Specifically, the Blaise 4 Windows software allows the code to be divided into components or modules that could be easily integrated together and removed apart again for further refinement. However, this life cycle model doesn't demonstrate the entire picture of what we do, so we invented a new model as presented in Figure 4.

Combination Incremental/Iterative Model

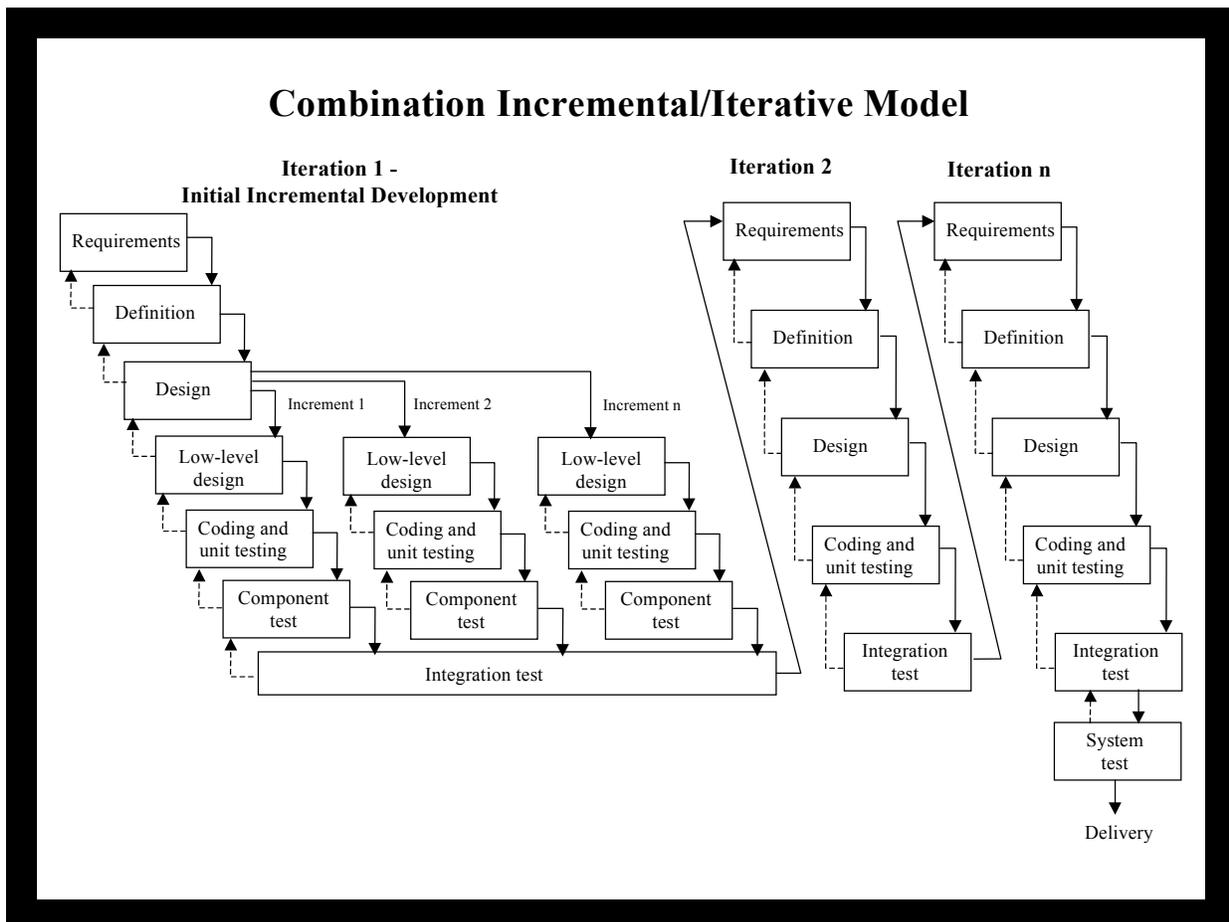


Figure 4

This combination of the incremental and iterative models of the software development life cycle allows for an initial development cycle that is incremental or partitioned into modules for development followed by multiple iterations of the classic waterfall model. Requirements are frozen once they are initially agreed upon. Once an overall design is completed, the development effort is divided into modules or increments so that lower level design, development, and testing can be done concurrently on the modules. The modules are then brought together at the end of the initial development effort to be integrated. Each time an iteration is completed, this model allows for a user test of a baseline product (i.e., Integration test on the diagram). The comments that result from the user test are added to the existing requirements to develop the new requirements for the next iteration.

This model allows for an initial modular and thereby, rapid approach to development followed by planned and controlled modifications to the system. This model is especially good for software development when the requirements are not completely defined up front because it allows for the discovery of new requirements as you go through the development cycle.

The combination software development model demonstrates clearly the work that is needed to adequately test instruments. Instrument testing is a very large and important task often not considered in developing project schedules. The TMO Authoring staff now includes this life cycle model in software development plans as a mechanism to help others understand the development and testing process.

How are instruments made available to the sponsors?

Software applications that are used by others, especially ones that have a graphical user interface, must be designed with the end user in mind. They should be intuitive and easy to use, be beneficial, and be designed so that they can be supported. To support the development, test and release of multiple instruments, the TMO Authoring Staff developed our own 'Tester's Menu'. The Tester's Menu is a system that runs on multiple platforms, i.e., Windows 2000, NT and 95, and allows users to test instruments developed in CASES and Blaise.

The primary responsibility of the TMO Authoring Staff is to develop instruments. The support and maintenance of the Tester's Menu is secondary. It was important to design the Tester's Menu to meet the basic needs of software developers and instrument testers. It was also designed so that it is easy to deliver, install, support and maintain.

The application driver is installed via e-mail using a program written in WinBatch, "The Batch Language for Windows" from Wilson WindowWare. The application resides on the server, whereas, the client's desktop receives only one executable necessary to provide the connection to the server. To change the functionality of a button or other menu function, an executable on the server side can be swapped without having to reinstall the application on every user's PC throughout the Census Bureau. The Tester's Menu system is being used at headquarters, regional offices, and the National Processing Center in Jeffersonville. Currently, the system supports several hundred users.

In an effort to make this menu as flexible as possible the program attaches to resources only when needed and lets them go when the user exits the application. It does not demand specific network or configuration modifications by other divisions, which makes this application easy for users to install. For Windows NT computers we use the set environment variable to drive DOS applications without having to modify the config.sys or autoexec.bat files on the user's PC. In fact, this software is so easy to use it installs the start icon on the user's desktop in all cases including dual boot systems.

Another important benefit of the design of the Tester's Menu system is the ability to dynamically make changes to the menu pick lists. A flat ASCII file is input to a stock Blaise manipula program eliminating the need to recompile source code to update the Blaise menu pick lists. The menu pick lists, e.g., survey name, are made from reading the directory structure. Adding, hiding or removing a directory makes an instrument available for use without additional programmer intervention, support or maintenance.

We answered the challenge of testing numerous instruments by building a testing system that is available for everyone to use. It provides a common ground for all testers of all instruments. The driver was programmed to have error-trapping built-in with useful messages given to the user if resources are unavailable or menu selections are not present. By keeping it simple to maintain and making it easy to install we build confidence that the products we make are reliable and useable.

TRAINING

Challenges

Training is an ongoing challenge. Designer, programmer and interviewer alike need access to training for several purposes: initially in how to do their jobs, to stay current with new developments in industry, and may even require re-training due to advancements in technology and/or survey methodology.

Our office is involved with finding new and better ways to introduce subject matter specialists, sponsors, project managers, programmers and field interviewers to different types of software design and implementation. Part of our task is to educate the sponsors and our staffs about how design decisions and coding techniques may adversely impact development issues such as instrument performance. Translating the sponsor's instrument design into a machine version, which can be used by the interviewers, is both challenging and rewarding for the authors. Part of the challenge is enabling good communication to share knowledge, skills and abilities. Education is mutually beneficial to all parties involved.

Developing an interface that has a common look and feel for our field interviewers that will also work with different CAI software is an important way to minimize training. Interviewers need to collect data quickly and efficiently. The development and implementation of standards allows the interviewer to learn the instrument faster, gain confidence in using the instrument, and adjust to using other instruments more easily.

Approaches

How is training addressed from a software development point of view?

Training is important during all phases of the instrument development process. Teaching programmers how to use the Blaise Control Centre and other tools is essential to accomplish the task of coding and building instruments. Educating our sponsors and subject matter staff on new methods and techniques is a wise investment. We also benefit by making sure the methods used to collect data across numerous types of instruments are consistent.

Having a well-documented and standard approach to interviewing simplifies the training of field interviewers. Standards for screen design and function keys assignments gives us the opportunity to create instruments that look and feel the same for field interviewers regardless of which interview is conducted. Training is simplified when the interviewer knows what to expect from survey to survey. Our challenge is to develop good CATI/CAPI Standards for GUI instruments. A GUI Screen Standards group was formed to develop, prototype, and user test screen standards for the Consumer Expenditure Survey. This was the first CAPI survey to be implemented using a Windows-based software language, Blaise 4 Windows. Decisions about screen colors, fonts, layout size, and function key assignments were made based on current and past experiences, prototyping efforts, performing usability tests with interviewers, and collaborating with other agencies that use Blaise 4 Windows.

Predefined standards for screen design go a long way toward minimizing unpleasant movement when navigating the instrument. By identifying the common elements we want to use across all surveys we can train the interviewers what to expect. The use of pull down menus or the speed bar and inclusion of some or all of the functionality of hot keys can be decided, defined and documented in advance simplifying not only the instrument design process, but also the training required.

Function key definitions are very much like the standards for screen design but on a smaller scale. Each instrument should have similar definitions so that interviewers can expect to use the same keys for the same functions across surveys. Every effort was made to keep keys the same between our CASES

instruments and the Blaise instruments. Many of the functions such as “exit”, “notes”, “help” are standard across surveys and should have consistent key mappings across surveys. Optional function keys have been set aside for special functions that do not apply across all surveys giving maximum flexibility.

The Blaise windows menu (.bwm) file gives us the ability to bind keys to meet our needs. With the freedom to easily define hot keys comes responsibility. It would be unwise to have every programmer and every survey sponsor make up their own key definitions. We have taken the approach of identifying keys used by CASES and those used by Blaise and reconciling the two to support a generic approach for all surveys built and supported by the Census Bureau.

Training issues are understandably part of our industry. As surveys migrate from one authoring language to another, as new computer based interviewing methods are developed, and as new developers, programmers, managers and field interviewers are hired they will need training. To simplify this task as much as possible we establish effective communication, develop and implement training opportunities and work to continually develop and enhance standards for CAI.

SUMMARY

The Technologies Management Office Authoring Staff are dedicated computer professionals tasked with developing methods and procedures that are generic and can be applied across all surveys, which make our Computer Assisted Interviewing instruments of better quality and more cost efficient. Our office supports standardization to the extent it is helpful in achieving The U.S. Census Bureau’s mission, “To be the preeminent collector and provider of timely, relevant, and quality data about the people and economy of the United States. “

Our staff acts as a human interface and turns ideas and subject matter specifications into procedures and programs for use by Computer Assisted Interviewing technologies. Our work requires us to interact with a host of other people, organizations and systems. The successes we have achieved would not be possible without our clients, sponsors, system developers, testers, trainers, and field interviewers.

Windows based design is well suited to modern machine capabilities and for a variety of techniques and features for data collection such as multi lingual text, voice, graphical and video presentations. With this array of opportunity naturally comes complexity. Our job is to meet the challenges of developing these complex instruments while balancing the sponsor wants, with what the software language and computing resources can do.

The approaches we are using to surmount these challenges have been and will continue to be incorporated for use in future survey automation efforts. We continue to address the issues and implement procedures to streamline project management, standardize instrument development, facilitate testing and improve training. Our objective is to continually improve our processes, while producing instruments that meet our sponsors’ needs on time and within budget.

Session 2: Programming Techniques

- Dynamically Created Answer Categories in Blaise Instruments
Carol Oppenheim, Battelle, USA
- Different Ways of Displaying Non-Latin Character Sets in Blaise
Leif Bochis, Statistics Denmark
- Programming Techniques for Complex Surveys in Blaise
David Altwater, Ventrese Stanford, and the Blaise Authoring Team
U. S. Bureau of the Census Technology Management Office
- Output Processing for Consumer Expenditure Survey
Xiaodong Guan and Howard R. McGowan
U. S. Bureau of the Census

Dynamically Created Answer Categories in Blaise Instruments

Carol Oppenheim, Battelle Memorial Institute

The use of enumerated answer types in Blaise data models provides great control over the data collected during the course of a study. These enumerated types have specific meaning for investigators and are an aid to analysis. One disadvantage of enumerated types is that they require a “one size fits all” approach. There can be circumstances within interviews that would make it desirable to have individualized responses instead of the same exact choices for all participants.

A way to take advantage of the best features of enumerated answer types and still present meaningful answer choices to respondents is to create answer categories dynamically during the course of each interview. Offering personalized answer choices that have specific meaning for each respondent is an excellent way to enhance interviews. For analysis, every person’s data will have responses stored in the same format but for administering the interview, each person will be given different options. By dynamically filling the response categories, it is possible to store a response with specific meaning for the researcher and present as choices responses with specific meaning for the respondents.

There are three steps in setting up an enumerated answer type in this way. First you must establish some identifiers that will be used as text fills. The fills can be set up as fields or as locals. Using fields for the fills allows the actual text that was presented to the respondent to be stored in the data set. If there is no need to store the actual text, locals work equally well.

The second step is to create answer types that will use the fills as answer text. Each response will have a name and text. The text will consist of double quotes surrounding a caret and the fill identifier, as in “^DayOne”. The answer type cannot be part of an included answer type file, but must be defined in the module(s) in which it will be used.

Finally there must be program logic to determine the content of the text fills. There are numerous ways to determine the text for the fills. Some examples are the interview date, whether the respondent is a case or control, or responses provided previously by the respondent.

At Battelle, we have successfully used this technique on a number of studies. It has been useful for questions with varying degrees of complexity. For some questions we knew that every respondent would be offered the same number of choices and that only the text content needed to be determined individually. In other cases, we needed to determine how many response choices each person needed, as well as the content of each choice. There were other instances in which the answer categories needed to be matched with particular ordinal values, a requirement that usually involved being able to have discontinuous numbering of the responses presented to any one respondent.

In a study of depression among adolescents, there is a series of questions for which the researcher wanted answers with the meanings “The current month”, “One month ago”, “Two months ago”, “Three months ago”, “Four months ago”, and “Five months ago.” Although the researchers were interested in the relative time periods, the respondents would be likely to answer with actual months. In order to properly select answers from these categories, the interviewer would need to use the respondent’s answers to calculate back from the current month, an error-prone process that would slow down the interview. Without dynamically created answer categories, the answers for this question would be presented to interviewers and respondents in the following format:

0. The current month
1. One month ago
2. Two months ago
3. Three months ago
4. Four months ago
5. Five months ago

We were asked to program those questions so the respondent could choose actual months based on the month in which the interview takes place. So for an interview being administered today, the answer choices would be September, August, July, June, May, and April. The text for these answers is filled as soon as the interview is started, using the month of the interview date as a point of reference. Looking at the code needed to accomplish this, we first established the following fields as part of the main source code for the data model:

ThisMon : STRING[9]

LastMon : STRING[9]

Back2Mon : STRING[9]

Back3Mon : STRING[9]

Back4Mon : STRING[9]

Back5Mon : STRING[9]

In this case, we chose to use fields instead of locals for the items to be filled because there were questions in more than one block of the interview that relied on these responses. Rules for determining the text to go with these fields were also written in the source code for all months of the year:

```
IF IntDt.MONTH = 1 THEN
  ThisMon := 'JANUARY'
  LastMon := 'DECEMBER'
  Back2Mon := 'NOVEMBER'
  Back3Mon := 'OCTOBER'
  Back4Mon := 'SEPTEMBER'
  Back5Mon := 'AUGUST'
....
ELSEIF IntDt.MONTH = 9 THEN
  ThisMon := 'SEPTEMBER'
  LastMon := 'AUGUST'
  Back2Mon := 'JULY'
  Back3Mon := 'JUNE'
  Back4Mon := 'MAY'
  Back5Mon := 'APRIL'
....
ENDIF
```

Within the interview, there were numerous questions that asked respondents in which of the last six months they participated in particular activities. For each of those questions, the answers that appeared on the screen were the fills created in the main source code. The respondents could choose the answers “September” and “July” and the interviewers could choose those exact responses from the options on the screen. The interviewers did not have to do any “translation” to know that July was two months ago. They selected “July” and it was stored as “two months ago” or, numerically, as code 2.

```
SET OF  
(CURRENT (0) "^ThisMon",  
Minus1 "^LastMon",  
Minus2 "^Back2Mon",  
Minus3 "^Back3Mon",  
Minus4 "^Back4Mon",  
Minus5 "^Back5Mon")
```

For respondents interviewed in September, this is the appearance of the enumerated type on the screen:

```
0. SEPTEMBER  
1. AUGUST  
2. JULY  
3. JUNE  
4. MAY  
5. APRIL
```

Comparing this to the fixed answer categories, we can see that these responses are easier to work with, both for the interviewer and the respondent. They may even help with respondent recall since the respondents won’t have to count up months in their heads to know what time period the questions refer to.

Sometimes we know that there is a maximum number of responses that we want to have available for all respondents but a smaller number for some individuals. In a study of services available to new mothers and their infants, one of the questions asked the mother when she came home from the hospital. Mothers were supposed to keep a diary of services used for up to 16 days beginning with the birth of their child. The baby’s date of birth was already known before the start of the interview, so we knew that was the earliest date on which the mother and child could have come home and on which the diary could begin. We did not want to continue the diary questions past the interview date or past the 16th day of the child’s life. In this study, the use of dynamically filled answer categories permitted us to display anywhere from one to 16 possible dates on which the mother could have come home. We filled the categories with precise dates, including the day of the week. The mother could reply with either an exact date or a day of the week and the interviewer would see both forms of the date on the screen. For those respondents whose answer list included fewer than 16 days, the unused days were not assigned any fill and they were not displayed on the screen. It is important to note that the unused days did still exist as answer choices, even though they had no text and were unseen. To prevent an interviewer from selecting an inappropriate response, we had to have logic that precluded their use. This was accomplished very easily with the following code that was repeated as needed for additional days.

Assuming the interview was taking place prior to the infant’s tenth day of life, the code for day 10 would have looked like this:

```
IF Day10 = EMPTY THEN  
    ComeHome <> Day_10  
    “The date you have selected is after the current date.”
```

ENDIF

The answer choices for a mother who gave birth on July 20 and was called on July 23 appeared as follows:

1. Friday, 7/20/2001
2. Saturday, 7/21/2001
3. Sunday, 7/22/2001
4. Monday, 7/23/2001

Without using the dynamically filled categories, it probably would have been necessary to have three separate fields in order to determine the date of homecoming. The first field would have been for interviewer use only (not to be read aloud) and would have been something like “Did R answer with a date or with a day of the week?” Depending on which answer the respondent provided, the program would then have to go either to a field for entering the day of the week or one for entering the actual date. Our dynamically filled answer categories eliminated the need for a lead question and displayed easily understandable dates instead of such answers as “Day of baby’s birth” and “Third day after bay’s birth.”

In a study of unintentional injuries, we needed different response choices for each respondent. These choices were based on how the respondent completed a roster of children in the household and on who else was part of the household. The roster was completed in detail on up to 10 children under age 15, including their first names. In addition we also collected the number of children ages 15 – 18, but no other details on those children. In every instance there was an adult respondent. There could also be other adults in the household. In one section of the interview, respondents were asked if anyone in the household had been bitten by a dog. The answer categories needed to include first names of all children under age 15, one category for children 15 – 18 if there were any present in the household, one category for the respondent, and one for other adults, if any were present in the household. The enumerated answer categories had values from 1 to 13. Categories 1 through 10 were always reserved for children under age 15. These were identified in the data as CHILD1, CHILD2, CHILD3, etc.

```
(CHILD1 "^CH1",  
CHILD2 "^CH2",  
CHILD3 "^CH3",  
CHILD4 "^CH4",  
CHILD5 "^CH5",  
CHILD6 "^CH6",  
CHILD7 "^CH7",  
CHILD8 "^CH8",  
CHILD9 "^CH9",  
CHILD10 "^CH10",  
CHILDOVER14 "^CH11",  
RESP "^RESPONDENT",  
OTHERADULT "^OTHADULT)
```

Once the makeup of the household had been determined, the answer categories could be filled appropriately. In a household with 3 children under age 15, one child at least 15 years old, and a respondent, the answer categories would be:

1. Freddy
2. Sue
3. Alice

11. Child over age 14
12. Respondent

Categories 4-10 and 13 would not be needed so they wouldn't be filled or displayed. As in the example above, these categories continue to exist and we still needed to write logic to prevent anyone from selecting these invalid choices. When the question about dog bites was asked, we then cleared the text from those categories that were not selected by the respondent. The code is, again, very simple:

```
IF NOT(Child1 in DogBiteQuestion) THEN
    CH1 := EMPTY
ENDIF
```

The same code was repeated for every possible response. If two household members were bitten, we only want the interviewers to select from those two people for subsequent questions related to dog bites. The new answer list, to be used for the question "Of the people bitten, who sought medical care?" might look like this:

2. Sue
12. Respondent

All other household members and all unused slots for household members continue to exist as possible but invalid answers. The same logic used previously to keep invalid choices out of the data is still needed here.

An example of using text fills for responses in a case-control study is a study in which the cases were part of a special program. As part of the program, children of cases received intervention from program specialists who worked out of pediatricians' offices. When asking cases and controls who at the office provided services, we would want everyone to be offered certain choices, such as doctor, nurse, nurse-practitioner. Only cases should be offered "program specialist" as a choice. For controls, the answers would be:

1. Doctor
2. Nurse
3. Nurse-practitioner
5. Social worker

For cases the answers would be:

1. Doctor
2. Nurse
3. Nurse-practitioner
4. Program specialist
5. Social worker

The answer type looked like this:

```
THelper =
(Doctor,
Nurse,
Nurse_p "Nurse-practitioner",
Spec "^Code4",
SocWork "Social worker")
```

The correct set of answers was determined for this study by a digit of the case ID that identified the respondent as either a case or control. The logic used to set the contents of the answer text was:

```
IF SUBSTRING(CASEID, 3,1) = '1' THEN
    CODE4 = 'Program specialist'
ELSE
    CODE4 = ''
ENDIF
```

Another way to have different choices for cases and controls is to set this up as two different questions, one that was only used for cases, the other for controls. Each would have its own answer type. When converting the data file to a format in which it could be delivered to the client it would be necessary to assign the answers to the two questions to the same data location. By using the dynamically filled answer categories, we already had the data from the two types of interviews in the same location. This also made the program clearer than it would be with two different fields being used for the same portion of the interview.

These examples are just a few of the ways in which dynamically filled answer categories can be incorporated into Blaise datamodels. As we use this technique, we continue to identify more and more situations where it will help us produce clearer interviews and better data. Using dynamically filled answer categories does require some additional programming up front, but the advantages make the extra effort very worthwhile.

Different Ways of Displaying Non-Latin Character Sets in Blaise

Leif Bochis, Statistics Denmark

Abstract

Blaise offers excellent support for multi-language interview instruments allowing the interviewer to select interview language on the fly. While this is easily managed with European languages using characters supported by the standard ANSI character set, there are a number of problems to address when using non-Latin character sets - for example, Arabic script.

This paper addresses certain problems concerning preparation of documents in non-Latin scripts and how to incorporate these as field texts in Blaise instruments - including problems of converting texts from word processing format into the simple text file format supported by Blaise.

As new versions of Blaise offer still more new facilities in general, also new ways to make non-Latin characters displayable in Blaise instruments is introduced - through separate display software, as field texts using special character sets and fonts or as images.

The aim of this paper is to describe a range of possible solutions and to discuss advantages and drawbacks of each solution in existing and (close) future versions of Blaise.

This paper elaborates further on a paper presented at the 6th Blaise Users' Conference in Kinsale 2000 and discusses advantages and drawbacks of the different solutions exemplified by Farsi text, based on work with incorporating Farsi field texts for Immigrant Surveys carried out by Statistics Denmark 1998-2001.

What is it all about?

Statistics Denmark has carried out a number of Immigrant surveys over the last 3 years concerning living conditions, integration on the labour market, language skills et cetera. Among them a major survey in 1998-99, which was repeated – slightly modified – in 2000-2001.

The questions concerning language skills implied that the respondents should be interviewed in Danish if possible, otherwise in their own language if applicable, or in English as a third alternative. The instrument should therefore be able to provide question texts in a range of different languages.¹

The surveys were designed as multi-language surveys and made use of the language facility of Blaise in the implementation in order to enable the interviewers to change language on the fly. The questionnaire was for the purpose translated from Danish into the languages English, Polish, Serbo-Croatian, Turkish, Somali, Arabic, Farsi, Urdu and Vietnamese.

Six of the languages are written with (a variant of) the Latin alphabet, which caused no trouble in the implementation, the latter four languages, however, are written with either Arabic script (Arabic, Farsi, Urdu) or a widened version of Latin (Vietnamese). The efforts made to get the texts incorporated in the Blaise instrument were concentrated on the Farsi version.

What is the Problem?

The solutions should be developed to work as CATI applications and run in the Danish (i.e. standard Western) version of Windows NT – through the period in which Blaise versions from Blaise III over Blaise 4.1 to Blaise 4.3 were upgraded.

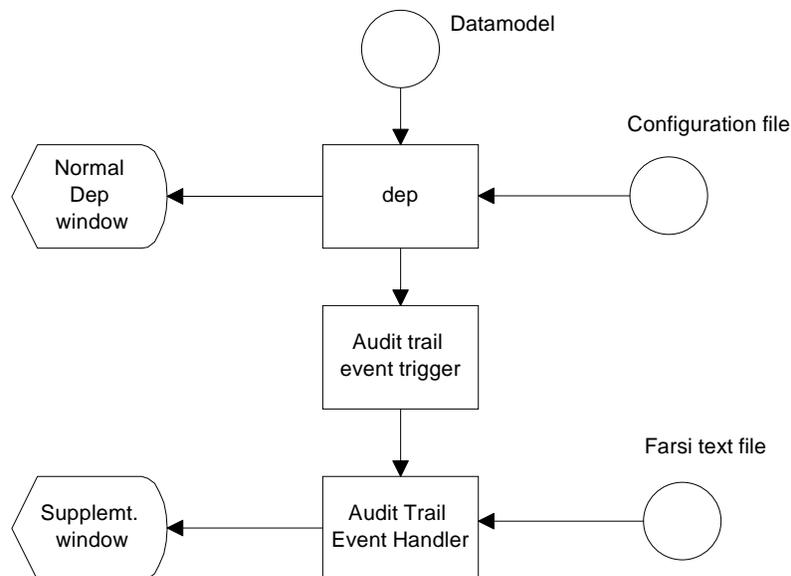
With the Dos version (Blaise III) the options were generally limited by the operating system. Some local versions of Dos supported an Ascii codepage including a local script – also with right to left orientation – which made it possible to switch between, for example, Latin and Arabic or Latin and Hebrew, but in practice it would hardly have been possible to develop Blaise-instruments using more than one non-Latin alphabet. At least, while we were running Windows NT as operating system, we experienced that it was difficult to apply additional codepages for Dos applications.

When Blaise entered the Windows world many more features were added allowing, for example, the use of more than one font – The early versions, however, were based on the OEM character set in order to keep compatibility between the Dos and Windows versions. To accomplish, this all data and metadata were stored in OEM format and automatically converted to ANSI when displayed in the Windows version, causing that the use of special fonts to display foreign alphabets were practically impossible.

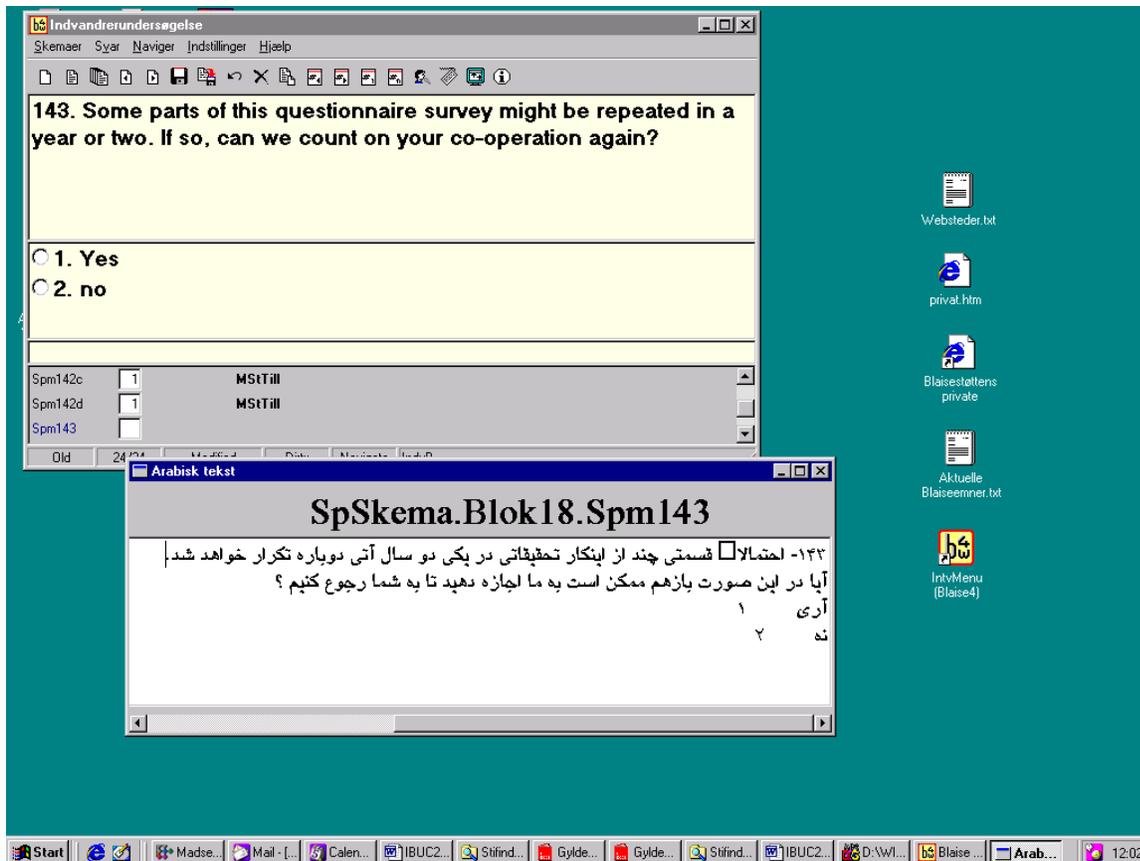
The use of Windows, however, made calls to external routines and programs far easier, and the introduction of the audit trail mechanism made it possible to let different kinds of displaying take place in separate systems but still controlled from Blaise.

Audit Trail Solution

The Blaise 4 Windows Audit Trail mechanism triggers the relevant events during the interview such as the AuditTrailEnterField event, which is triggered each time a field is entered and passes the relevant information (Field Name, for example) to the Audit Trail Event Handler.



For the 1998-survey a special version of the Audit Trail Event Handler was developed in order to manage the communication between the Blaise instrument and a supplementary display window. This Audit Trail Event Handler then looks up the proper question text and displays it through the Supplementary Window Control.



This Audit Trail Event Handler was written through a slight modification of the Audit Trail example, delivered with the Blaise system and is described in detail in ref. 1. This system was made ready in the last phase of the 1998-survey and worked reasonably well in testing, though it wasn't used in production.

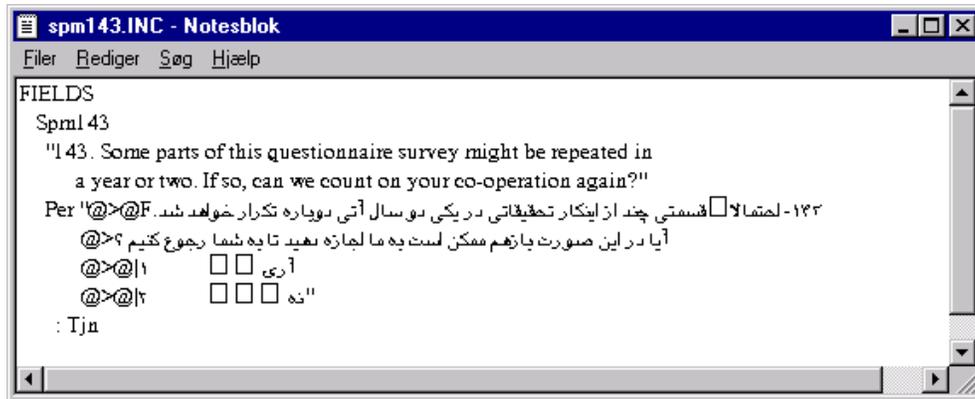
Blaise 4 Ansi-solution

With version 4.3 Blaise became ANSI-based, and oem-ansi conversions were no longer needed. Thus, it became possible to use the built-in facility to define special fonts to use in (parts of) field texts – for example, to represent other scripts than Latin – provided they could be represented by a single-byte character set.

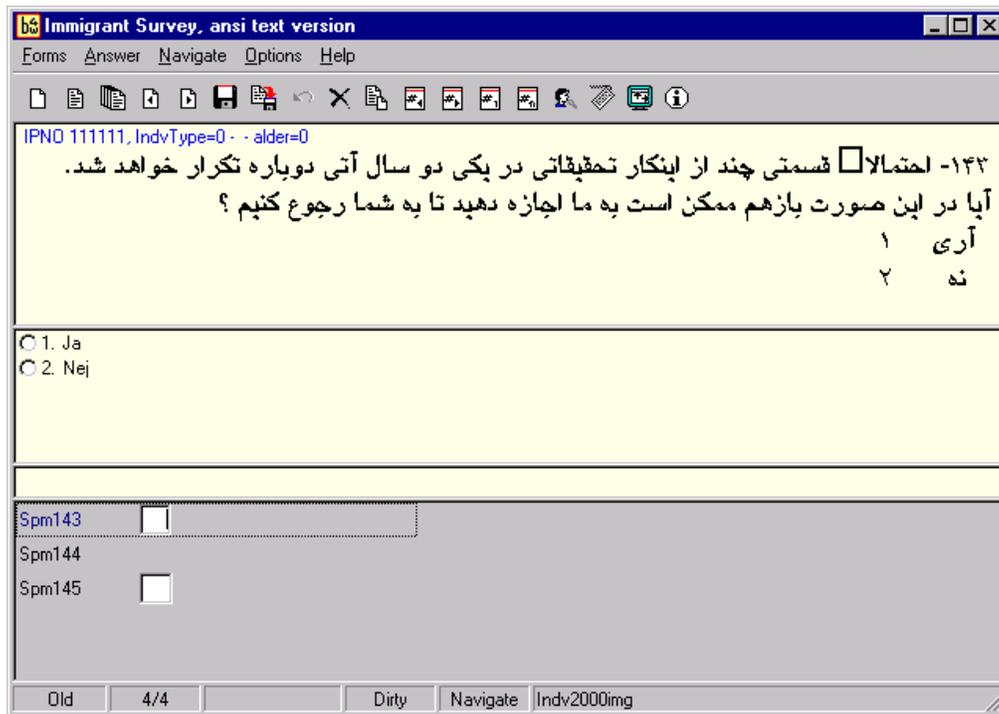
Certain considerations, however, should to be addressed when, for example, Arabic script needs to be displayed in Blaise Field texts.

Arabic is written from right to left (hereafter referenced *orientation*), is aligned to the right (*alignment*) and the shape of the letters is dependent of where the letter is positioned in the word (*presentation form*).

These properties are not handled by the Blaise Editor, and consequently the translators will need a word processing tool to work out the translation and afterwards there will be a need to convert the outcome into a plain text format that may be inserted into the Blaise datamodel source.



@F here denotes the use of Farsi (Persian) Font, while @>, @| and hard spaces take care of alignment and spacing for better readability.



As the Blaise field text display cannot by itself handle orientation, alignment, presentation forms and line shifts for Arabic text, it is important that all these properties are generated during the conversion process.

All this, however, implies that it is very important to state a set of requirements concerning the format of the delivered translations in order to achieve that the format is actually either possible to apply directly in the Blaise source, or that it is possible to convert for that purpose.

Sparse attempts based on the delivered Farsi text from the 1998-survey showed that this should be possible to carry out in practice in the next survey. So in 1999 we were quite optimistic and thought that "next time we are ready!"

Discourse: Why We Were Not Ready!

What complicated this approach, however, is the nature of a constantly changing world. What worked fine as a conversion program two years ago didn't work in 2001 because there was no program to produce the same kind of output. Though the same person who did the 1998-translation should also do the 2000-translation of the questionnaire, the Farsi Word processor that was used in 1998 was not available. Instead, Microsoft Word 2000 was used to produce a Farsi version of the revised questionnaire – primarily on the basis of a plain text file that was used in the Audit Trail solution for the 1998-version.

Word 2000 is actually capable of processing Farsi text, i.e. Word 'knows' what part of a certain document that is in Farsi, English or whatever of a wide range of supported languages with their respective scripts. The letters are stored in the Unicode double byte character set and displaying as well as printing is controlled by Word concerning orientation, default alignment and actual presentation form.

The translator therefore decided to use Word to write his translation and passed the resulting Word document on for further processing.

In order to pass it over to a plain text format that could be inserted in Blaise field texts, the Word document was converted to a Unicode text file, which in turn could be converted into a plain text file with a single byte character set (with respect to orientation, presentation forms etc.) that might be displayed with a True Type font that could be used in Blaise field texts.

Unfortunately, the delivered Word document for a greater part consisted of the translation from the 1998-version, which was copied into Word from a plain text file and – supplied with the proper font and alignment – looked rather good in display and print. Word, however, was not 'aware' that it actually was Farsi text but 'thought' it was Danish with respect to Unicode-representation, orientation etc. The resulting document turned out to be quite a mix of 'Farsi' and 'quasi-Farsi' text.

Though we were ready to receive a Farsi text and had selected a suitable font for the purpose, we didn't manage to get the received text to fit into the text format expected.

Imaging

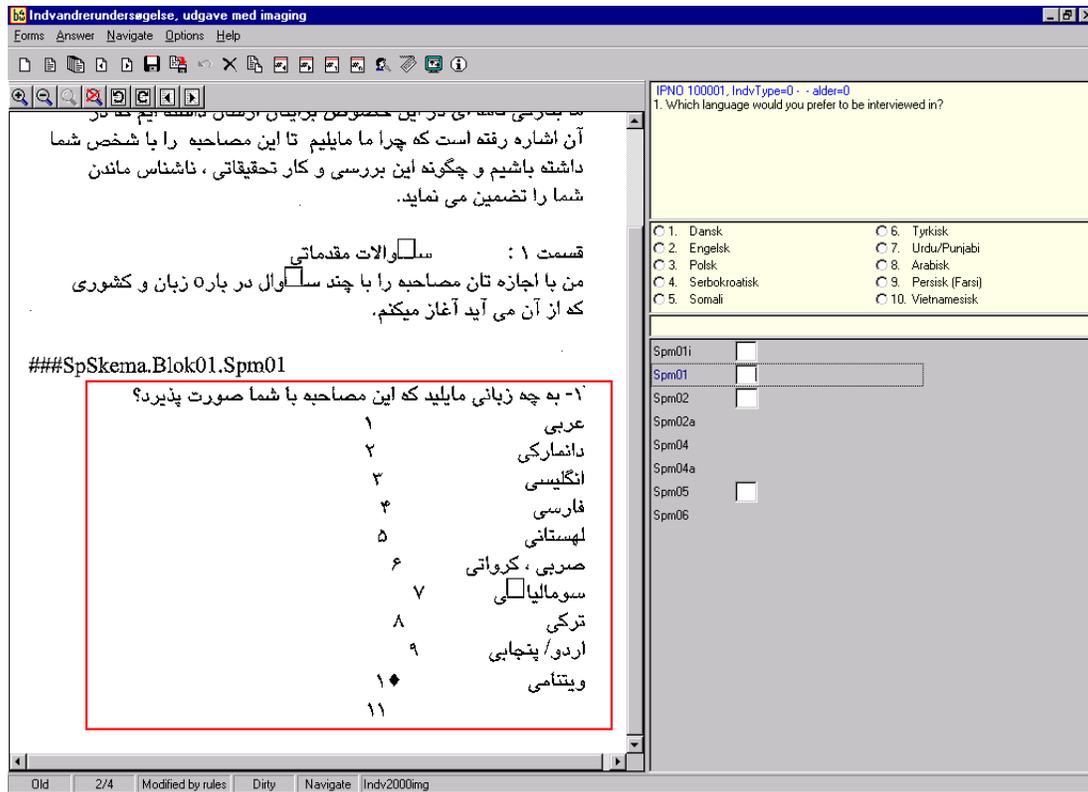
With Blaise 4.5 and BCP a quite new way to get across the conversion and editing problems were introduced – the imaging feature. As explained in the documentation this feature is originally intended for displaying scanned paper forms in a data editing process in order to ease data editing - and probably reduce the amount of paper and burden of paper handling.

This feature makes possible a general way to show graphics on the screen and at the same time relates parts of the graphics to specific fields of a datamodel. In this way it is possible to show field texts

(eventually combined with graphics) in any script of the world - provided only that it may be presented on plain paper. And – most important of all – it can be done

- Without the need to bother how the script is represented on the screen
- Without writing or purchasing third party software to handle the presentation
- Without the need to specify any file format in which the translators should deliver, and
- Without needing to convert from one file format to another.

You might even make a template that defines how much size should be used for each field (because a long field text in Farsi probably also would be a long field text in Chinese, Arabic or any other language) – and thus make a general solution for all language versions involved in the survey.



It is not possible, however, to change to a quite different language (for example, from Farsi to Chinese) right on the fly and continue interviewing, as the graphics files are read by the program upon each entry of a new form. You will need to redefine the respondents preferred language, make an appointment and save the form. When the appointment comes forward next time, the images will then show the Chinese version – but probably another interviewer is needed also in this situation.

Take an example:

- The interviewer calls the respondent, and finds out that though an immigrant from Iran the respondent appears to be Chinese and wants to be interviewed in Chinese.
- The interviewer fills in the answer that the preferred language is Chinese.
- The program then fills in the proper values – names of files with the scanned texts – in the Imaging Management fields.

- The interviewer makes an appointment, and transfers to a Chinese-speaking interviewer.

Other solutions

An alternative graphical representation should be mentioned. It is possible – and has been since the first Windows version of Blaise – to display graphics also as bitmaps in the field pane of a Blaise instrument through the use of a multi-media language. This requires the preparation of individual bitmaps for each question and with a large questionnaire that could be quite a tedious job. For that reason, we haven't followed that trail in our search for possible solutions. On the other hand, there is a number of advantages of the solution. Because the name of the bitmap files may be defined in run time, it is possible to change the non-Latin-based language right on the fly. This also allows the preparation of slightly different versions of the questions, although that implies an increase of the preparatory work to be done.

Conclusions

With the newer versions of Blaise, still more features are added allowing new ways to work out solutions to problems concerning how to provide question texts in non-Latin scripts.

First, the Audit trail mechanism in Blaise 4.0 – 4.2 allowing events during interviewing to control other programs, and thus make it possible to control displaying from Blaise that Blaise cannot display by itself. The solution suffers from the fact that text substitution in field texts cannot be done unless you are willing to develop tailored Audit Trail Event Handlers for each instrument that should be deployed. On the other hand, it allows the use of third-party software and standards like Unicode, which might ease the preparatory work.

Second, the ANSI-based versions 4.3 – 4.4 with the possibility to define special fonts for other scripts (without the need to worry about oem-ansi conversions) and then make it possible to display any single byte character set, that can be presented through a True Type font.

And third, the imaging feature of Blaise 4.5 and BCP that makes it possible to display the text as graphics alongside the ordinary field text.

The Imaging solution also suffers from the fact that text substitution in field texts is not possible – on the other hand, the solution supports that the ordinary field text is displayed alongside and here supplies the interviewer with any additional information.

If text-substitution in field texts is needed there is no realistic alternative to the built-in language facility – this facility, however, is only well suited for languages written with the Latin alphabet (and is probably too problematic to apply in practice for a multi-language instrument).

Both the Audit Trail solution and the inclusion of non-Latin alphabets in the Blaise field texts suffer from the fact that the conversion from a given script into a suitable simple text file format – and eventually later editing – might be quite a cumbersome project, though the Audit Trail solution does open up more possibilities, for example, the non-Latin text does not have to fit into a single byte character set.

The imaging feature of Blaise 4.5 / BCP is without competition the fastest and easiest way to incorporate non-Latin scripts in a Blaise questionnaire as almost all concerns about conversion and presentation can be avoided.

In the future – as Internationalization emerges – coming versions of Windows will probably support the use of non-Latin scripts better – for example, through the consistent use of Unicode and support of controls able to display Unicode-characters and take care of matters like orientation, alignment, presentations forms etc. And Blaise eventually might support Unicode controls for field text displaying.

As long as this is not the current state-of-art, other solutions must be applied. Blaise still offers a variety of ways to do this.

References

Leif Bochis: *Audit Trails or How to Display Arabic Text in Blaise*, in: Proceedings of the 6th International Blaise Users' Conference, Kinsale 2000, available at www.blaiseusers.org

Blaise documentation, *Audit Trail*, in: Blaise 4.1 Developers' Guide, Chapter 5.6 Audit Trail, pp. 282-293, *Showing Form Images in the DEP*, in: On line documentation for Blaise 4.5

ⁱ Please note that it has only been the ambition to provide question texts in non-Latin scripts. It is not the intention of this paper to discuss the possibility of changing the whole user interface, including typing in answers to questions in other languages than what is default in Statistics Denmark.

Programming Techniques for Complex Surveys in Blaise

Ventrese Stanford, US Census Bureau TMO Authoring Staff

David Altvater, US Census Bureau TMO Authoring Staff

Curtis Ziesing, US Census Bureau TMO Authoring Staff

Software organizations are constantly seeking strategies to improve their ability to develop and deliver high quality software in a timely and efficient manner. The U.S. Census Bureau's Technology Management Office (TMO) Authoring Staff is implementing the Capability Maturity Model (CMM)¹ as a strategy to improve their software development process. The CMM identifies five levels of software process maturity, from the initial chaotic stages of development (Level 1) to an optimized, mature process (characterized by Level 2 through Level 5). This paper focuses on some of the software tools the Census Bureau's Authoring Staff developed as part of the strategy to support the Level 2 CMM methodology. The tools are used as a strategy to establish continuous process improvement. By establishing both the software techniques and the methodology to effectively manage those tools in future projects, we support the Level 2 CMM objective - Repeatability. Effective and efficient programming techniques are identified and documented and will serve, with other process improvement tools, as the basis for developing reference materials for new projects. These successful techniques are proposed as a standard across Blaise instruments developed by the Census Bureau.

The Census Bureau recently added the Blaise 4 Windows Software to their box of programming tools for automating surveys. In converting to this language from the DOS based CASES language that has long been the standard in the Census Bureau, many prototypes were developed to test complex data collection requirements in the Blaise environment. This presentation will present some of these complex prototypes and other challenging requirements that will serve as the basis for the Census Bureau's computer assisted interview (CAI) standards through a discussion of some of the screen and internal design strategies. These designs support the practice of repeating successful practices in future automation survey development projects. And finally, the discussion concludes with some insight from some of the lessons learned and suggestions for future Blaise software enhancements.

Screen Design

The objective in the screen standard design is to provide an interviewing environment that is meaningful, familiar over time and responsive to the multiple needs of interviewers during data collection. This objective has resulted in standardizing features ranging from the use of various function keys to the color and size of screen text. Every attempt is made to develop an interviewing environment that is efficient with a look and feel that is similar across Blaise surveys. Also considered in the Blaise design are CASES design elements. As the Census Bureau's makes the DOS to Windows conversion in surveys, some design elements have been carried forward from the DOS based CASES surveys to minimize interviewer confusion. For example, when developing standard Function Key number assignments in Blaise, whenever possible the assignments commonly used in CASES surveys were adopted.

Starting with the basic "out of the box" Blaise default screen displays, described below are some standards that have been established for the text, color, interviewer instructions, navigation guides, reference information and errors. Implicit in the design is the effort to clearly display as much information as needed on one screen.

¹ *The Capability Maturity Model: Guidelines for Improving the Software Process*, Carnegie Mellon University, Software Engineering Institute, 2000, Addison-Wesley Longman, Inc. Boston, Ma.

Book and Diamond Symbols

Since “a picture is worth a thousand words,” symbols are used wherever possible as instructions to the interviewer. In Figure 1 below, the book symbol indicates to the interviewer that they should display page 13 of a bounded information booklet for the respondent to review. The diamond symbol indicates that the following text is not to be read but is an instruction to the interviewer. The book and diamond are created using a True Type Wingdings font. The character display is set to char (38) for the book and char (115) for the diamond. The following code shows the assignment:

```
AuxBook      := '@Z'+Char(38)+'@Z'  
AuxDiamond  := '@Z'+Char(115)+'@Z'
```

Auxbook and AuxDiamonds are string auxfields. The @Z is a user defined font in the modelib that has been assigned the True TypeWingdings font. When the book or diamond is needed to display in the question text area, the parameter is passed to the block and referenced as a variable in the question text. The following code shows the call to the block with the parameter passing and the actual field question text setup that displays the symbols.

```
Section1(AuxBook, AuxDiamond)  
  
Block Section1 “Called Block”  
  Parameters Import AuxBook, AuxDiamond : String  
  Fields  
    Field1 “^AuxBook @/@/ ^AuxDiamond Question text”  
  Endblock
```

Windows uses this type of visual identifier by creating the icon and Blaise displays the icon using the ^ command. The @/@/ indicates to Blaise to advance two lines down in the screen. The result of this technique is a small picture the user can clearly and quickly identify as specific instructions for the current question.

Status and Speed Bars

The standard screen display information available has been increased to give the maximum space real estate. For example, the Blaise Speed Bar was eliminated. The speed bar was removed because the Census Bureau laptop systems are focussed on keyboard data entry. The speed bar is a GUI feature that relies on the use of the mouse. The same graphic user interface (GUI) functionality is offered elsewhere on the keyboard. This elimination resulted in some space savings on the display screen. Also, the flexibility of Blaise enabled the development of standard information to display on the Status Bar, such as case identification number, date, interview number, respondent name, block page number, and field name for reporting problems.

Font Type, Color and Size

Font type color and size are used consistently to guide the interviewer in reading text. The Census Bureau has established GUI screen standards of font type size and color. For surveys developed in a graphic environment the Bold Times Roman font size 12 text indicates to the interviewer information that is read to the respondent, blue text are instructions to the interviewer, and gray text are optional text that are read to the respondent only when necessary. Figure 1 below illustrates this standard. Grey text usually occurs when collecting data for repetitive lists. The text is displayed in bold black the first time it

is read. After the respondent's first response, the text is displayed in gray indicating to the interviewer that the leading phrase in the question is read only if necessary for continuity and clarity.

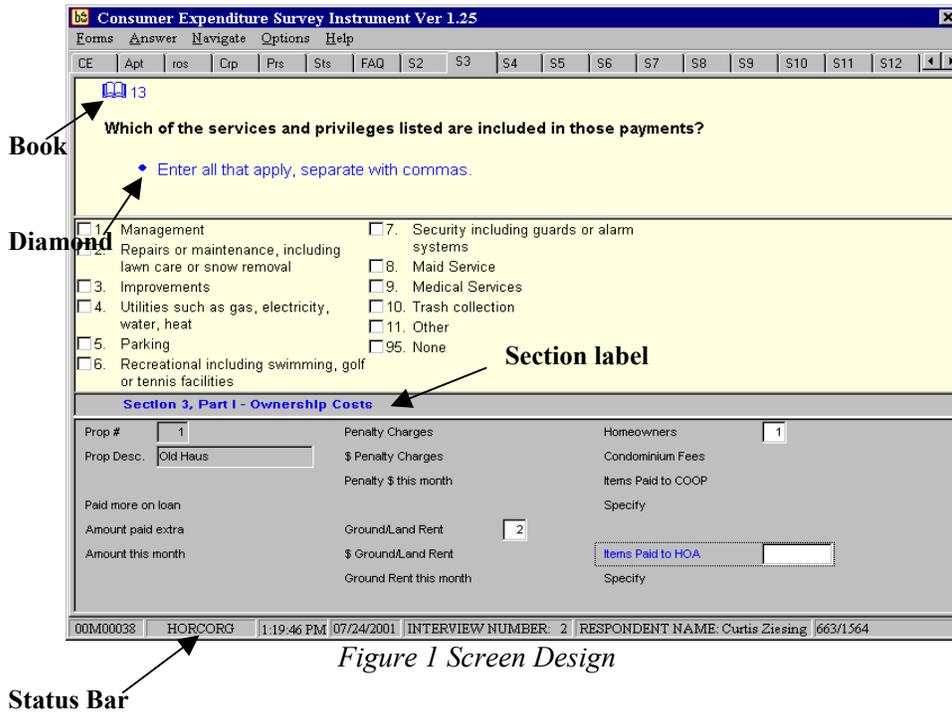


Figure 1 Screen Design

The code below, illustrates how the black text changed to dark gray in Figure 2. In this example, if the first item in the laundry list has been answered yes and there are more questions, then the flag LTGrey which is an auxfield is assigned the value yes. When the LTGrey flag equals yes and there are more questions then the auxfield DKGray is populated with the user defined font @K. The @K has the value for the font size, the color gray and its text type.

{Ask the screening question and set the flag}

```
Item
IF Item = NoMore Then
  LTGrey := No
ELSE
  LTGrey := Yes
Endif
```

{If the flag is set and it is not the last question then make the text gray}

```
IF LTGrey = Yes AND NOT ITEM = NoMore THEN
  DKGray := '@K'
ELSE
  DKGray := ""
ENDIF
```

{Assign the appropriate color in the question text}

```
Question := DKGray + 'Since the first of last month have you paid for -' + DKGray +
 '@' + ' Read each item on list.'
```

{Ask the question}

```
Question.ask
```

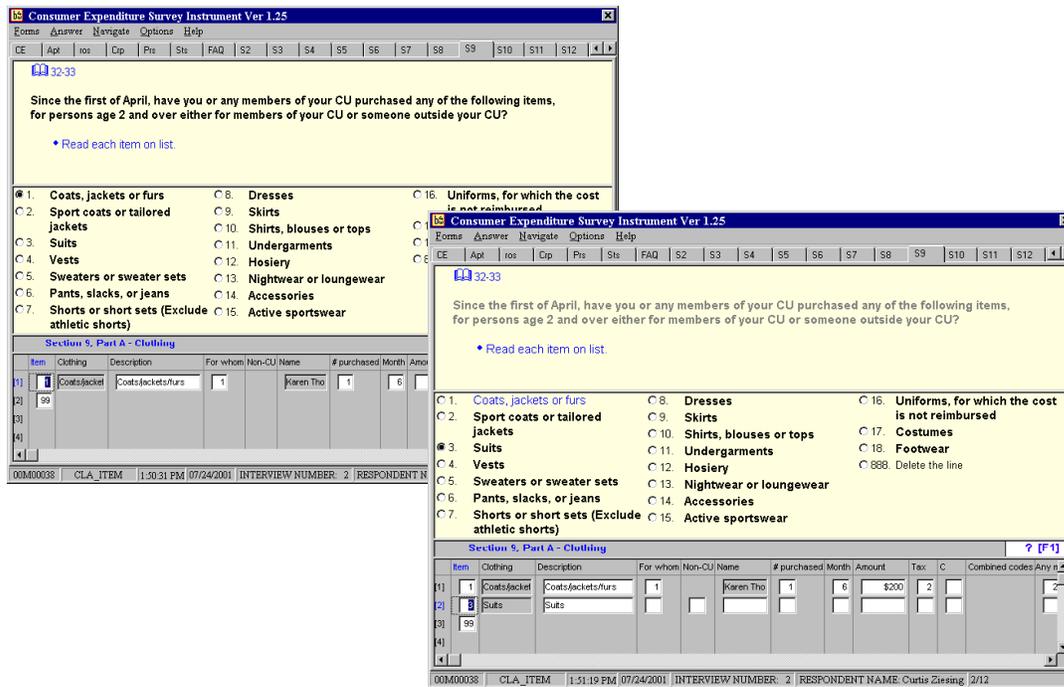


Figure 2 - Laundry list with the Text Bold and Then Gray

Section Labels

The Consumer Expenditure Survey (CE) has 23 major sections with questions on topics such as housing, clothing, and cars. Many of these sections contain subparts that focus on questions in a very specific area such as the housing section having subparts for mortgages and home equity loans. Blaise enables the interviewer to skip around to various questions during the interview. This random navigation results in the need for the screen to have distinct and clear identification of each section and part. This specification was addressed with the use of bitmaps as shown in Figure 3. These bitmaps were created using Microsoft Windows Paint although any software could be used. The bitmaps were then integrated at the field level in Blaise. For questions that have links to specific help screens, a second bitmap label was created adding the ?[F1] symbol. The ?[F1] symbol on the right of the screen informs the interviewer that the same information is also available on a help screen for them to follow in support of a particular question by simply pressing the F1 function key. Displaying these symbols result in an efficient use of the limited space available on the screen.



Figure 3 – Example of an Individual Section Label Bitmap

Code at the Field Level with the Bitmap Image

Fields

ITEM “Since the first of April, have you or any members of your CU purchased any of the following

Items for persons age 2 and over either for members of your CU or someone outside your CU?" "image(info09aH.bmp, TOP=345, LEFT=0)" : TSec9aItems

The code above shows the bitmap link including its location on the screen for the field ITEM, it is 345 points down from the top of the screen and starting 0 points from the left of the screen. The code was standardized in the illustration for placement of this bitmap by assigning the value to an auxfield. This shortcut enables easy revision for all fields in the block sharing the same screen layout.

Fields

ITEM "Since the first of April, have you or any members of your CU purchased any of the following Items for persons age 2 and over either for members of your CU or someone outside your CU?" "^TEMPIMAGE": TSec9aItems

Rules

TEMPIMAGE:= 'image(info09aH.bmp, TOP=345, LEFT=0)'

Help Files

As previously described, the ?[F1] symbol indicates that a help screen is available. This screen is linked to the F1 function key. The help file is created as an external file linked in Blaise by using the DEP menu manager tool. The help file in this example was created using Microsoft Help Workshop but it could be developed in any help software program. The text can be written in any editor that supports rtf. Once the rtf file is created, it is compiled in a help format. The compiled file must have a .hlp extension for the Blaise link through the DEP menu manager as shown in Figure 4.

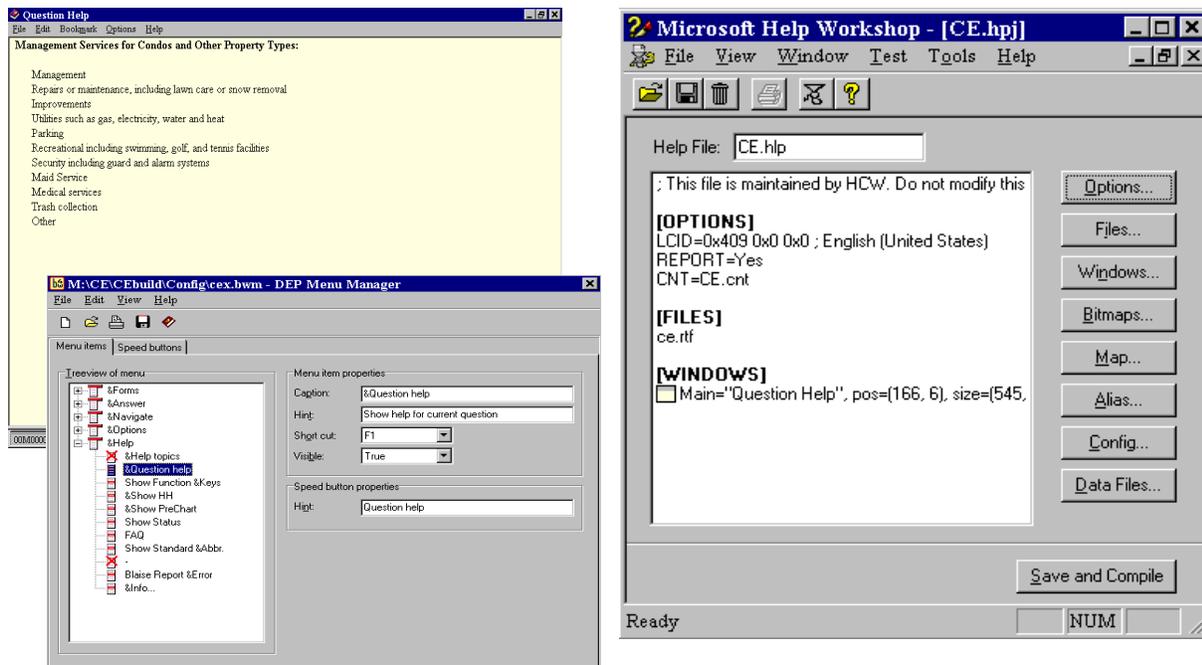


Figure 4 – Help Files

Error messages

The specifications for the CE survey required numerous edit checks for valid data ranges. Identified were six common error messages used in the checks. These messages were standardized and declared at the data model level. The messages are passed as parameters into various section blocks as needed. This method enabled an easy establishment of a standard in appearance and text across all sections in the survey that can be easily edited. The consistency enables interviewers to quickly interpret the problem and address its resolution.



Figure 5

The code below is an illustration of how the error messages can be standardized at the datamodel level. The error messages are assigned to an auxfield at the datamodel and then passed into various blocks requiring these specific types of messages. The message is displayed in the block upon entering an error. This technique enables easy editing of the messages and a standard continuity throughout the instrument.

Datamodel

```
Err1 := Diamond + 'THE VALUE ENTERED IS UNUSUALLY HIGH OR LOW' +  
      Diamond + 'PLEASE VERIFY'  
Err2 := Diamond + '100% WAS ENTERED' +  
      Diamond + 'PLEASE VERIFY'  
Err3 := Diamond + 'MONTH ENTERED IS NOT IN REFERENCE PERIOD' +  
      Diamond + 'PLEASE VERIFY'  
Err4 := Diamond + '$0 WAS ENTERED FOR EXPENSE' +  
      Diamond + 'PLEASE VERIFY'
```

Section03 (Book, Diamond, Err1, Err4)

Block BSection3

Parameters

Import

Book, Diamond, Err1, Err4 : STRING

Rules

SPECIALX

IF SPECIALX = RESPONSE THEN

SIGNAL

SPECIALX >= 8 AND SPECIALX <= 22400 "^Err1"

ENDIF

Internal Design

The focus of the internal design for this paper is developing modules of code or methods of data collection within the software instrument which are reusable either within the same instrument or by other instruments, a repeatable methodology. Surveys in the TMO Authoring branch are designed considering the requirement of repeatable code that can be used throughout the instrument or in other instruments. Several examples of these designs are presented such as the laundry list in the Consumer Expenditures Survey, the open grid and the current status block developed in the American Community Survey's Telephone Follow-up project. This section concludes with some internal design strategies to improve instrument performance.

Laundry Lists

The 'laundry list prototype' was developed in conjunction with Mark Pierzchala of Westat. It is a set of modules used to collect expenditure data about a range of grouped and similar expenses on selected items. The structure of this prototype provides standard, reusable modules of code that are easily revised. These modules are used throughout the Consumer Expenditures Survey instrument with slight modifications for each instance. An advantage of the laundry list prototype is that the interviewer is presented with the same basic screen appearance while collecting different types of data. This feature enhances continuity and speed in the interview. In designing this prototype, the basic premise was to have a survey-programming concept that would allow the collection of any type of data even though it is segmented into specific parts. Examples of these parts are vehicles, medical payments, education expenses, clothing, and appliances. Each part is further segmented into various groups to collect more specific data. For example, Figure 6 shows Section 6, Part B of the appliance laundry list in the CE Survey. To distinguish between parts, a code of "95" is used as a separator. The first group within Part B is for small appliances. The next group is for larger appliances. The final group is for general equipment. When all similar items are completed the "95" is entered to indicate that the group is complete. The interviewer can then begin entering data for the next group. Once the data is collected for the final group, the interviewer enters a "99" to indicate the completion of the part.

Figure 6 - Laundry List

In terms of the internal design of a laundry list, a generic shareable table layout was used. Collecting data in a table allows the user to visually access information as it is entered into the survey form. This method enabled the table to be ‘plugged’ into any section of the instrument that asked questions in a laundry list style. The table block for this instrument is called ‘TTable’. By creating one generic shareable TTable, as modifications are required their results are reflected across all sections that use the laundry list. The technical side to the laundry list begins with a setup of the table itself. A field of type table would be defined. Within the definition of the table, there is a block of data specifically called BRow. Following that is the generic TTable code that has a field Row defined as type Brow as in the following snippet of code:

```
Datamodel Example
USES LLMetaData
Table BTableA {Blaise setup for rows and column input}
  Block Brow {each row would have many fields and error trapping}
  ... {Parameters and other code would be here}
  EXTERNALS LMD : LLMetaData
  ... {Other code would be here for locals and such}
  Type
  TSec6aItems = (Item1          (1) “^TempItem[1]”,
                ... {Additional selectable items would be here}
                Item45         (45) “^TempItem[45]”
                NoMore95       (95) “^TEMP95”,
                NoMore99       (99) “^TEMP99”,
                DeleteLine     (888) “^DeleteLine”)

  Fields
  Item : TSec6aItems
  ... {Many other fields defined here}

Rules
  IF LMD.SEARCH (SecNum, SecLetter) THEN
    LMD.READ
  ENDIF
  ... {A lot of rules code would be here}
Endblock {Brow}

Fields
  Row : ARRAY [1..100] OF Brow
Endtable {Ttablea}

Fields
TableA : BTableA
Endmodel {Example}
```

Within the block BRow there are some specific codes for the laundry list that are generic for all sections. This code controls the selection of “95” or “99”. It also controls the change of an entered item by

deleting the associated data if an item code were changed. There are many other error trapping routines to ensure the best and most accurate data is collected.

Another concept developed in conjunction with Mark Pierzchala of Westat is the use of metadata. This contains the text for question fills and selectable items and range edits for the CE instrument. It is a form of dependent data. The external metadata was designed in Blaise as a table. Each row in the table represents a section and its associated parts. For example, Section 6 and its two parts are identified by two rows in the metadata table. Once the external file was built, in the Blaise instrument at the data model level a USES statement establishes the metadata file for the interview. Within the block where the metadata file is used, the EXTERNALS clause is coded and then the actual code to call the file are the SEARCH and READ methods. Because this is general across all sections that use the laundry list, a generic section can be developed in a short amount of time, then modifications to cater to the actual section can be made. This allows focus to be placed on the data being collected and not so much on the method of collection. This technique also facilitates training, both programming and interviewing. Once the laundry list methodology is grasped it is leveraged throughout the instrument. The metadata sets the number of selectable items. So if there were 18 items listed in the metadata then only the first 18 items are filled. The challenge in this technique is to control the display of desirable items on the screen where there is less than the maximum in the array. Section 6, Part B has less than 45 items which results in displaying the empty array positions with selectable numbers and scrolling as displayed in Figure 7. To resolve this problem, the Blaise hide empty option was used, but the empty options could still be selected. An additional error trapping routine was developed to disallow the selection of empty items.

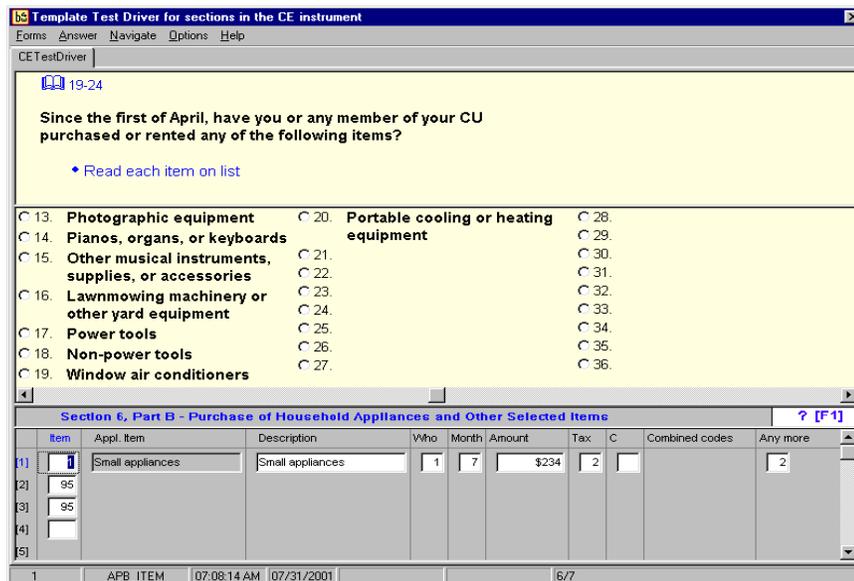


Figure 7 – Hide Empties Not Enabled

Once the item has been selected, the text fields in the metadata are transferred to the text fills in the instrument. If the item changes, the question text will change to fit the current item. Each section and part can input as many as six text strings that can be used as fills in the instrument. This can be expanded and it shows the versatility of Blaise. Each time an item is selected the data from these string fields in the metadata are transferred to the instrument fields and displayed. The text fills are not stored in the code, or hard coded therefore text can be modified independently after the instrument has been prepared. If one of the fields in the row requires a specific edit range based on the item selected then the metadata will

provide the high and low values for the edit check. For each item code, a different edit range is established through the use of the metadata file. Instead of having a long case statement, coding is reduced to using only one edit check and an error routine for any item code selected. A generic maximum number of 100 rows are used because each group needs to limit the number of rows collected. A variable called maxrows represents the limit for the number of rows collected for any given type. For some groups such as vehicles, a maximum of 25 items may be sufficient for collecting data; however for clothing, an entire household may require a higher maximum. The maximum number of rows collected using the TTable needs to be limited to preserve instrument performance.

Designing a screen that displays all data for each part on the screen simultaneously can be very challenging when there are a large number of items. The amount of time to develop these visual aesthetics needs to be considered when planning project time lines. The result of these visual aesthetics is a smooth flowing instrument that conveys information to the interviewer in the most convenient manner and enhances efficiency. Font colors were also standardized in the laundry list prototype to enhance the flow of the interview. When an item has been selected and recorded in the table, the text of the selected item is changed from black to blue. This gives the interviewer a visual marker that the item has already been selected. As previously described in screen design, the question text is 'grayed' when it is optional to read aloud to the respondent.

Open Grids

An open grid provides interviewers the flexibility to enter data in either a person based columnar style or in a topic based row style using a table. This method was developed in conjunction with Westat for the American Community Survey's Telephone Follow-Up instrument. This method as illustrated in Figure 9, was developed out of the need to provide the interviewer with the option to enter data either across a row or down a column and yet not slow down the data entry process. The data entry table structure enables the application of specific edits at the field level and the postponement of other error check edits until after the table is indicated as completed by the interviewer.

At the field level, edits are performed for selected fields such as age, e.g. where for example a value of 150 will trigger an immediate edit. Edits that are postponed until the interviewer indicates that they are finished collecting data for the table would be checks for empty fields. Using the EMPTY attribute on fields within the table allows the interviewer the flexibility of bypassing on route fields in a row and skipping around to other fields in the grid. At the point when the interviewer considers the table grid is complete, a "yes" is entered for finished and control is transferred to a block that performs edits on selected fields within the grid. If an edit is fired, the interviewer is returned to the field involved in the edit on the table. The edits are performed and data is entered or corrected on a field by field basis. Once the data is 'cleaned up,' control is transferred out of the table.

Current Status Block

In the American Community Survey Telephone Follow-Up project, a method was needed to determine the status of the entire data form upon demand. While flags could be set in the block to determine the status, the problem was that the results of the edit flags could not be tallied directly from the blocks where the flags were set. This is because the Blaise rules determine whether a field is 'on route' or 'on path' at any point in time. It is possible that a field that is currently 'off route' would not be included in the tally if the tally is computed at the field level. The current status block was developed in conjunction with Westat to address accurately calculating the field pass/fail flags at any point throughout the interview. This block calculates only the values of the flags and posts the error tallies by level of criticality and block identifier. The field based status flags in the instrument are constantly updated in the Current Status block. Since they are isolated, the RULES for all of the fields in all of the blocks are not

run in order to calculate the totals. The range of the flag's error status as shown in Figure 9 includes: low, medium, critical or no errors (implied by zero totals in the categories). This internal design of a dedicated block also improved performance in the instrument.

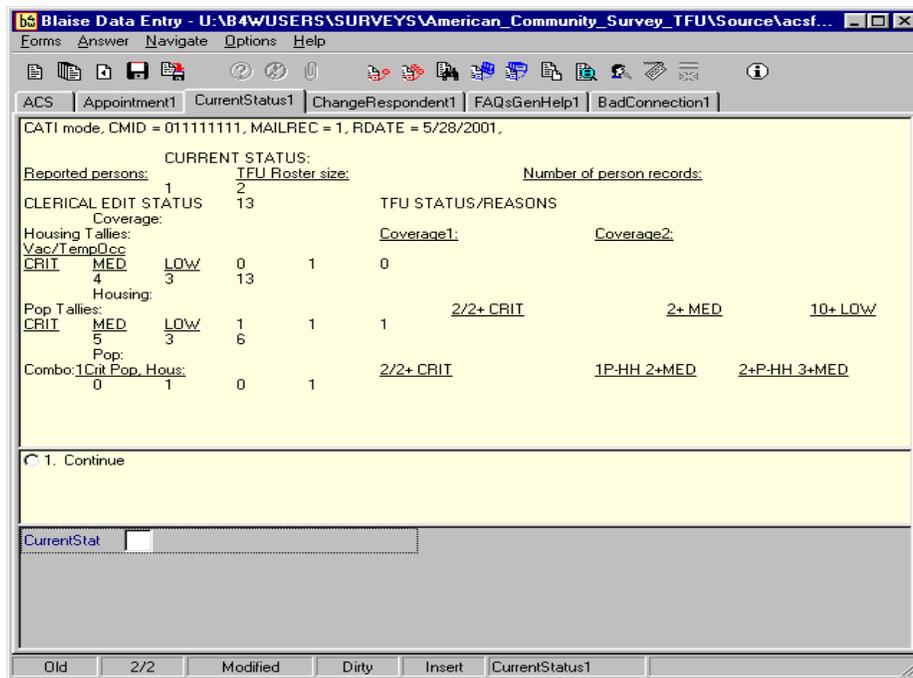


Figure 9

Performance Enhancement Strategies

Considering the configuration of the development platform, deploying Blaise instruments can require a large amount of cache memory. One of the most important requirements of interviewers is that they are able to collect data quickly especially when interviewing impatient and reluctant respondents. This requirement considers not just the length of the questionnaire but the ease of use of the instrument and its processing performance. Some survey instruments require many features and enhancements because of complex data collection issues. Extensive requirements can contribute to overhead processing which reduces the performance of the instrument. These requirements and the large memory demands of GUI based instruments drive the demand for high performing computers. A constant struggle in any technical organization is maintaining the most advanced technology available. This is especially true in the government sector where the acquisition of large numbers of computers can take many months and even years. The reality of most organizations is that the technology actually used will not always be the very latest. The Census Bureau fields over 6,000 laptops. Constantly maintaining the very latest technology is simply not cost-efficient. As a result, it is exigent that the programmer authors design instruments in the most efficient manner possible to maximize the use of the available technology. This means going beyond the basic programmer efficiency strategies such as avoiding long if-statements, or placing the most common condition first in long if-else statements, etc.

In going beyond these basic strategies, Blaise has some features to assist in enhancing performance such as running Cameleon to determine generated parameters and thereby minimize their use in coding. There is also the Watch Window blocking checking feature. This feature enables the authors to verify if only the appropriate blocks are being checked on selected fields. The Consumer Expenditure Survey project is currently the largest project in the Census Bureau that has been developed

in Blaise. The completed instrument contains over 360,000 fields. Addressing performance in an instrument of this size of has been especially challenging and has demanded even more performance enhancement strategies. Constant testing was done during development to identify effective strategies for improving the instrument's performance. These strategies include:

- Controlling the running of the Blaise rules. This has been one of the key strategies to improving performance in this instrument. This was accomplished by:
 - Making separate blocks for the fields and rules in multi-shared blocks. This separation enables more flexibility for when rules are run for specific sets of fields.
 - Applying 'gates' to large blocks of code. The gate requires the interviewer to put a block or section 'on route' only when needed as shown the code below:

Fields

Intro "Now I am going to ask about the purchase or rental of household items."
: (Continue (1) "Enter 1 to continue", NODK, NORF)

Rules

```
IF (Intro = Continue ) THEN
  {**** [ laundry list sections ] ****}
  Sect06
  Sect07
  Sect08
  Sect09
ENDIF
```

- Structuring survey sections and arrays that minimize pagination to improve performance in the initial setup process.
- Consider performance impact when using external files such as edit range lookups.
- Use LOCALS only for looping.
- Set any external files to read-only whenever possible.
- Avoid KEEP statements for entire blocks when possible.

Recently in performance testing and benchmarking, a significant performance refrain was found in the CE instrument. For the current laptop configuration that will be deployed in production, the performance threshold was significantly impaired when the instrument exceeded 240,000 fields. This prompted the authors to do some internal redesigning and explore alternatives. Currently being considered is the implementation of two separate instruments. This was possible because of the unique character of the CE survey. One instrument could contain a read-only database that holds dependent data from previous interviews for the main instrument.

Lessons Learned

While there have been many lessons learned from the surveys that were authored by the TMO authoring staff only several key items can be delineated in this limited document:

- High level requirements should be carefully reviewed as they relate to the structures of the language that will be used to implement the requirements in the early stages of a project. This enables appropriate designing of an instrument to support required functionality considering the constraints of the software as well as efficiency and performance in the instrument. At this stage repeating structures can be identified which can lead to reusable or standard applications within the context of the survey.

- The identification of structures to refine with prototypes greatly reduces the risk of creating inefficient data structures. Prototyping design methodology identifies potential problems in the proposed design at the early stages of development.
- Established standards should be reviewed throughout the process so that their use and implementation is ensured. Standards that conflict with functionality or known limitations should be renegotiated without sacrificing the goals of the users.
- A high level flowchart is essential to good development design. It helps document and identify intended functionality. Often reusable or standard structures (blocks) can be identified.

In summary, obtaining extensive documentation early in the planning stages of the project and assisting the sponsor in the survey design is most conducive to good programming practices, design, and implementation that ultimately produces the best and most accurate data collection instrument.

Suggestions for Future Blaise Releases

Some of the design strategies discussed in this paper could be included as features in future Blaise releases. Listed below are some suggestions for development of future Blaise software releases.

- Section labels and column headings including options for font type, size and colors should be included as an option when defining grids in the modelib. Also, more rich text format options for the description area of the grid as well as in other places. This would eliminate the need for section bitmaps.
- A feature that evaluates the status of on-path and off-path data. This is especially useful when trying to control the run of the Blaise rules or to tally the status of flags as described in the current status block strategy.
- Expand the edittype feature to enable the field to be passed as a parameter.
- Make the autohide feature a default setting. Currently empty items are displayed as illustrated in Figure 7.
- Enhance the Blaise editor with features for: file comparisons, customized formatting options on selected code such as bold, italics, color, etc.
- A command to perform KEEPS on *all* elements of an array simultaneously.
- A dialog box during compilation that notifies the developer of:
 - New or modified parallel block text descriptions in the .bxi.
 - New or modified types in a previously generated .bxi.
 - The masking format on newly added types in the .bxi.

As more surveys are developed in the Blaise programming language the demand for these features and other will grow but the current functionality of Blaise is exceedingly flexible and enables considerable creativity in developing instruments that collect quality data.

Acknowledgements

Special thanks to Mark Pierzchala (Westat) for his consultation expertise and prototype assistance; Roberto Picha (Census Bureau) for his testing and performance enhancements; William Dyer, Jr. (Census Bureau) for his benchmarking, and all of the TMO Blaise Authors (Census Bureau) for their fine development work that led to many of the examples and conclusions presented in this paper.

Output Processing for Consumer Expenditure Survey

Xiaodong Guan, U.S. Census Bureau
Howard R McGowan, U.S. Census Bureau

Introduction

The Consumer Expenditure (CE) Survey is a large and complex survey conducted by the U.S. Census Bureau for the Bureau of Labor Statistics (BLS). The survey is being converted from paper to Computer-Assisted Personal Interview (CAPI) in April 2003. A CAPI instrument has been developed using Blaise and several field tests conducted. From the paper survey, the Census Bureau currently delivers 68 SAS data sets containing approximately 10,000 variables to the BLS. The deliverable from the CAPI output will be similar to the paper output.

The CE data model contains hundreds of blocks and thousands of fields. There are many types of data relationships in the CE data model. Additionally, the block structure and field names are constantly changed as the instrument is developed, tested and modified. The data output requires that we generate the SAS data sets for the selected blocks and fields from the CE data model. The data export system needs to be easily modified for changes in either the data model or output requirements.

We considered three output options.

- Use the Export ASCII data file option

The Blaise system provides some tools to export data directly from the data model. Due to the size of the CE data model, the Export ASCII data file option does not work for the CE data model.

- Develop a custom-built Manipula output program for the CE data model

Such a program would have to be written block by block, on a field by field basis and would contain hundreds or thousands of lines of code. It would be hard to develop and test. It also would be difficult to maintain.

- Generate ASCII relational data files for the data model and create SAS code for each ASCII data file.

Hundreds of data files will be output from the CE data model. Unfortunately, the Blaise system does not come with a totally automated way to generate SAS data sets for the selected blocks and fields from the CE data model.

To accomplish the output requirement for the CE data model, we applied the third data output option. We developed an export system for the CE data model. We modified Cameleon scripts and employed an output control file concept. The system is driven by an output control file developed and maintained by subject matter analysts. The system is flexible, efficient, and easy to maintain.

In this paper we describe in more detail the CE data model and data output requirements. We discuss and compare the data output methods in the Blaise system, and explain the reasons for our choice of method. We present the customized Cameleon scripts developed for the export system. We describe the concept and use of an output control file to select output blocks and fields. The paper concludes with a detailed description of the integrated data export system.

CE data model and data export requirement

Output processing is not a simple task for the CE CAPI survey data. It involves many aspects of the survey project, including the CE data model, output requirements, and other survey operation related issues. It is very important to choose the right tools to output data from the CE data model. In order to build a good data export processing system, it is critical that the developer understand every aspect of output processing. The following provides an overview of the CE data model and output requirement.

- **The CE data model**

The CE data model was developed with Blaise software. It is a large and complex data model. It contains 270 blocks and 10,105 block instances. The CE data model produces over 250 ASCII relational output files. There is a total of 368,695 data fields with a total length of 13,853,483 bytes defined in the model. There are many different data types and relationships in the data model. The technical description for the CE data model is shown in tables 1 and 2. During the development stage of the data model, the block structure, block names and field names are constantly changed; for example, there were 255 blocks and 5,215 uniquely defined fields in the version 23 data model. But there were 296 blocks and 5,955 uniquely defined fields in the version 22 data model. Since the CE survey is a continuous survey, the data model will be released to the Field for data collection monthly. The block structure and field definition could be modified every month, especially, in the early stages of data collection. Because of this reality, it requires the data export system to handle a large, complex, and dynamic data model.

Table 1. Block and data fields statistics in the CE data model

Overall Counts	Value
Number of uniquely defined fields *1	5,215
Number of elementary fields *2	4,960
Number of defined data fields *3	368,695
Number of defined block fields *4	255
Number of defined blocks	270
Number of embedded blocks	14
Number of block instances	10,105
Number of key fields	1
Number of defined answer categories	1,207
Total length of string fields	13,079,645
Total length of open fields	0
Total length of field texts	191,668
Total length of value texts	56,187
Number of stored signals and checks	128,486
Total number of signals and checks	128,486

*1) All the fields defined in the FIELDS section

*2) All the fields defined in the FIELDS section, which are not of type BLOCK

*3) Number of fields in the data files (an array counts for more than one)

*4) Number of fields of type BLOCK

Table 2. Data Type information in the CE data model

Data Fields	Number	Length
Integer	147,339	635,940
Real	1,920	9,920
Enumerated	74,020	118,127
Set	5,500	9,555
Classification	0	0
Datatype	34	272
Timetype	3	24
String	139,879	13,079,645
Open	0	0
Total in data model	368,695	13,853,483

- **Output requirements**

The data output requires the creation of about 70 SAS data sets, which contain over 1,800 variables, from the CE data model. All output data come from 90 blocks in the current data model. About 30 of the SAS data sets can be output from a single block. Other data sets need to be merged or manipulated from two or more blocks. Since the block and field name are not unique in the CE data model, the same block or field name can be used in many different sections. For example, the block “**brow**” and field “**amount**” are used in 20 different sections. Each one of the blocks produces a separate SAS data set. The field “**amount**” goes to different SAS data sets with a new SAS variable name. For instrument development reasons, the field names in the CE data model are often different than the SAS output variable names. For the purpose of data editing and analysis, the requirements may add or remove some variables from the current output. The CE output also requires exporting the timer information for each section in the data model. The data export system must be very flexible and efficient to handle the output requirements.

From the CE data model and output requirements, we see that the data export system will deal with a large, complex, and dynamic data model. It not only needs to generate the SAS data sets for selected blocks and fields from over 200 blocks and 5500 fields but also must easily handle changes either in data model or output requirements. Since the data output method is the cornerstone of the data export system, we will review the output method in the next section.

The method selected for processing CE output

The Blaise software provides three ways to output data from the data model in Version 4.4. The simplest and easiest way is the ASCII export. The ASCII export outputs data from the Blaise data model to an ASCII data file. This method is good for small sized data models. The length of the CE data model is 13,853,483 bytes. This method can not handle such a large data model. The CE data model is a highly structured data model. There are a lot of array blocks in the model. This method would not be suitable for the CE data model even if the method could handle the size of the CE data model.

The second way is to develop a customized Manipula output program. As [2] described, a custom-built Manipula output program must be done block by block, on a field by field basis and will contain hundreds

or thousands of lines of code. It is hard to develop, test and maintain this kind of program. Such custom-built output programs are less efficient during the output processing. It needs more time to connect the CE data model and the data output models. For example, it takes 16 minutes to output 300 cases from the CE data model to a special designed data output model. It only needs 3 minutes to output those 300 cases on the same desktop if we use the next method.

The third option is the ASCIIRELATIONAL export. This method can generate separate ASCII data files (ASCII relational data files), based on the data model block structure. For each ASCII relational data file, the Blaise system provides the Cameleon script to generate the data model associated with the ASCII file. Furthermore, the Blaise system can generate the SAS code based on the data model. As we described before, the CE data model is a large and complex data model. The CE data model is also highly structured, and the block structures are well defined to suit the data output requirements during the specifications developing phase. The third output option, export ASCII RELATIONAL data set, fits the output requirement of the CE data model compared to the other two methods. It gives us the basic steps to export the data from the CE data model. That is:

- Generate ASCII relational data sets from the CE database.
- Generate the SAS code for each data set.
- Run the SAS programs to create SAS data sets.

However, the output requires us to generate SAS data sets from the subset of blocks and fields in the CE data model. The export system can easily handle the changes in data model and output requirements. Unfortunately, the Blaise software does not come with a totally automated way to generate SAS data sets for selected ASCII relational data sets. There is considerable handwork involved if we use this method to export data from the CE data model. After we reviewed the Cameleon scripts provided by the Blaise software, the scripts did not provide enough information for the output in the CE data model. The output format of the scripts needed to be modified for an automatic data export system. Therefore, customized Cameleon scripts needed to be developed for our data export system.

The Cameleon scripts for the data export of the CE data model

The Blaise system provides many examples of Cameleon scripts. These examples can help users to generate a data model and create the code or script for other software. They also can obtain the information about the data model, such as field name and block number. These scripts needed to be modified to fit the data export requirement in the CE data model. For example, the Cameleon script outputs all data models for ASCII relational data sets to a big file. If we want to use some of the data models, we have to open the file, find the models, and cut and paste the models to separate files. For a large data model, such as the CE data model, it will be difficult and error prone work. For the output of the CE data model, we developed customized scripts, **ceasciirel**, **cedic** and **cesas**, which are modified from the scripts **asciirel**, **dic**, and **sas**. We modified these scripts to:

- Add some functions in the scripts to accomplish the requirement of the CE data model, such as, output more blocks or field related information for the CE data model.
- Remove some functions that are not needed for the output to reduce the chance to have errors during the batch processing.

The script **ceasciirel**, like **asciirel**, generates data models for ASCII relational data sets. It outputs each data model to a separate file. **Ceasciirel** also outputs the block information for the ASCII relational data sets. It connects the block name, block number, data model name, and ASCII data set name. The information can help us to generate batch files during the output processing. The following shows the details of the block information. From the information, we can easily determine the data model and ASCII

data set. If we need to create a SAS data set for block 26, we just use the script to generate SAS code for a26.bla with ceq_alp1.a26 as an input file.

ASCII relational block information

Sub Data Model	Block Name	Block Number	ASCII Data Set
A26.bla	Btabx	26	CEQ_alp1.a26
A71.bla	Bsect3i	71	CEQ_alp1.a71
A211.bla	Bback	211	CEQ_alp1.c11

The **ceasciirel** provides the block information we need in the output processing. The script **cedic** generates the field-related information in the CE data model. The output of script **dic** lists every defined field (array field is counted more than once) in the data model. It is not suitable for the ASCII relational export method. The output is too large to use for the CE data model. The **cedic** only outputs uniquely defined fields. For each field, **cedic** outputs the field name, full name and block number for the field. The subject matter experts can use the information to identify the output fields. Since the output of **cedic** and **ceasciirel** include the block number, it will help us to identify the block to be output. For example, if CORRPRG is an output field, we can easily find out the sub-data model and ASCII data set for this field through the block number 71. An example of field information is:

Field information

Field name	Block Number	Full Name
CORRPRG	71	Sect03.prop.sect3i.corprg
Loantype	58	Sect03.sect3a1.curprop.loantype
Name	42	cc.unit.preson.name
Comb	95	Sect06.tablea.row.comb

The script **sas**, provided by the Blaise system, generates the SAS code for the data model. It was developed for older versions of SAS software. The lengths of variable name are limited to 8 characters. The field name is used as the SAS variable name in the data model. For the CE data output, the export system should be able to pick the SAS variable name different from the field name. The **cesas**, modified from **sas**, adds these functions. The **cesas** also removes the **proc format** commands because the output of the CE data model does not need this function.

The three customized Cameleon scripts not only generate data model and SAS code for each ASCII relational data file but also provide more block and field information for the CE data model. This information is fundamental to developing the data export system. It becomes possible to output select blocks and fields automatically.

The output control file for the data export of the CE data model

There are about 270 data blocks and over 5,000 unique fields in the CE data model. Only 90 blocks and 1,800 fields will be output from the CE data model. It is very difficult to select the output blocks and fields by hand. We needed to find a way to select output blocks and fields automatically. The subject matter experts have developed a database to manage the CE project. The database includes much useful information for output processing. For example, it identifies all the fields that need to be output. The output file name (SAS data set name), is assigned to each output field name. For each output field in the database, if we can automatically identify its location in the CE data model, such as, full name or block number in the CE data model, it will help us to select the output blocks and fields. The field information produced by the script **cedic** can help us to accomplish this task. After merging the field information with

the control database, the subject matter experts can produce a data output control file or output specification. It contains the output field name, SAS data set name, and full name for each output field. An example of the control file is shown in table 3.

Table 3. Example of data output control file

Blaise Field Name	SAS Data Set Name	Fullname1	Fullname2
Name	MEMB	cc.person.name	
Age	MEMB	Cc.person.age	
Apadesc	APA	Sect06.table.row.apadesc	Sect06.table.prechart.row.apadesc
Apaitem	APA	Sect06.table.row.apaitem	Sect06.table.prechart.row.apaitem

Since the field name is not unique in the data model, there is more than one full name for some field names. From the output control file, we can produce two lists:

- **Output field list**

For each output SAS data set, the SAS variables are listed in the output control file. We can create a variable keep list in the SAS code for every SAS data set. When the SAS data set is created, we only keep the variables in the variable keep list.

- **Output block list**

First, we merge the output control file and the field information produced by script **cedic**. A new list is produced. The list includes field name, SAS data set name, block number for the field, and full name. (If the field is defined in more than one block in the data model, both block numbers will be listed in the file). The new list looks like:

Field	SAS Dataset Name	Block Number	Full Name
Name	MEMB	42	Cc.person.name
Age	MEMB	42	cc.person.age
Apadesc	APA	95	Sect06.table.row.apadesc
Apadesc	APA	94	Sect06.table.prechart.apadesc
Apaitem	APA	95	Sect06.table.row.apaitem
Apaitem	APA	94	Sect06.table.prechart.apaitem

Second, we combine the above list with the block information produced by **ceasciirel**. It will create a list that contains the SAS data set name, ASCII relational data set name and data model name. We create an output block list. An example is:

SAS Data Set Name	Block Number	Sub Data Model	ASCII Data Set Name
MEMB	42	A42.bla	CEQ_alp1.a42
APA	95	A95.bla	CEQ_alp1.a95
APA	94	A94.bla	CEQ_alp1.a94
APB	95	A95.bla	CEQ_alp1.a95

From the output block list, we see the data for SAS data set MEMB come from block 42. The data for APA come from blocks 94 and 95. The data in block 95 will go to different data sets. For each block in the list, we can use script **cesas** to generate SAS code and create SAS data sets. The processing can be done automatically if a batch file is created.

The output control file, generated from the CE project database, is easy to update if there are any changes in the data model. For example, when a new CE data model is released, a subject matter expert can automatically combine the new output from **cedic** and their database to produce a new output control file. If anything is changed in the output requirement, the database will be updated. The output control file will include the new change. Thus, the data export system is driven by the output control file. This allows the export system to be flexible, efficient and easy to maintain.

The data export system of the CE data model

Implementing all the parts we discussed above to an export system is not just putting everything together. The order in which tasks are performed has a big impact on the efficiency of the system. Keeping the necessary middle processing information may reduce time to repeat running the whole system. We had to consider all different possibilities during the development of the export system. The following are what we have considered or experienced while developing the data export system:

- **Perform tasks between SAS and Blaise**

The data export of the CE data model not only deals with a large, complex data model and special output requirements but also involves two data processing software, SAS and Blaise. Many tasks can be performed in either system. For any given task, we have to decide which software is best suited to perform that task. For example, we have two ways to keep the selected output fields. One way is to only keep selected fields when generating data models for the ASCII relational data sets. The advantage of this way is the system only carries the necessary data during processing. It reduces the size of files since the CE data model contains much duplicated information for display purposes. Some fields are extremely long. The disadvantage is that it requires us to develop more specific manipula programs or Cameleon scripts to accomplish it and needs more processing steps. Another way is to create a variable list of selected fields. When the SAS data set is created, we only want to keep selected fields and drop the rest of the fields. The unnecessary fields are kept until the last step. This is the easiest and simplest way to perform this task. After comparing the two methods during an early pretest, we chose to implement the second method for our export system.

Some tasks can be performed in either software without much difference. We generally use SAS to perform those tasks because we have more experience in SAS programming.

- **Select method based on efficiency in the system**

The Blaise system provides two ways to output ASCII relational data files. One is to output all ASCII relational blocks and another way is to output only selected blocks. In the output processing for the first two CE tests, we only output the selected blocks. We found that there is no big difference in performance between outputting selected blocks and all blocks. However, it took more time to process the data when we wanted to add more selected blocks later. We decided that output all blocks in the first step of the data export system.

Based on the control file and the customized Cameleon scripts, we developed the data export system for the CE data model. The system has two parts. The first part generates the SAS code for the selected output blocks and fields. This part can be executed every time to output the CE data. It may be executed

once and used for a while if the data model and output requirements are not changed. The export system includes the following steps:

- Generate field list from the data model.
- Generate the block list and data models associated with the ASCII relational data sets.
- Generate selected output field and block lists.
- Generate the batch files from the control list.
- Generate SAS code associated with the selected ASCII relational data model.

The programs, which are used and generated in this part, are listed in the table 4.

Table 4. The programs used or generated in the data export system

Step	Program name	Purpose	Input	Output
1	Cedic.cif	Generate the field related information for the CE data model	The CE data model	An ASCII file, CEDIC , that includes all uniquely field name, full name. The block number for each field is also in the file.
2	Ceasciirel.cif	Generate data model for each ASCII data set and block information.	The CE data model	Over 200 Blaise data models
3	Cecontrol.sas	Create the output field and block lists	CEDIC and a control file provided by subject matter experts	Output field list for each SAS data set. Output block list, Blocklist , for selected blocks. Ceoutput.sas
4	Ce4batch.man	Create batch files to generate .bmi, .sas and sas data set in output block list	Blocklist	Bla2bmi.bat Bmi2sas.bat Cesas.sas
5	Bla2bmi.bat	Prepare the selected data model to generate *.bmi files	The selected data model, such as, a98.bla	*.bmi files for the selected data models (e.g., a98.bmi)
6	Bmi2sas.bat	Generate SAS code for each selected data model	*.bmi files for selected data model	*.sas file for selected data model, (e.g., a98.sas)

The second part exports the data from the CE data model and generates the SAS data sets. The steps are:

- Generate ASCII relational data sets from the CE data model.
- Generate the SAS data sets for the selected blocks.
- Generate the final CE output.
- Generate timing information for each section in the CE data model.

The files used for this part are listed in table 5.

Table 5. The files used or generated in the data export system

Step	Program name	Purpose	Input	Output
1	Ceascii.man	Generate ASCII relational data sets for the CE data model	The CE data collecting database and data model	Over 200 ASCII data files (e.g., CEQ.a98, CEQ.a99)
2	CESAS.sas	Output the SAS data sets for selected data model	The selected ASCII relational data file (e.g., CEQ.a98)	SAS data sets for selected data file (e.g., a98, a99)
3	Ceoutput.sas	Create SAS data set for the CE output	The SAS data set for selected output blocks (e.g., a98, a99)	SAS data set for the CE output (e.g., APA, APB, etc.)
4	Ceaduit.sas	Creat a SAS data set for the time information.	The audit trail file for each case	A SAS data set contains the time information for each section in the CE data model

A **Winbatch** program links all the programs in the data export system. The system achieves our goal of exporting data from the CE data model and generating SAS data sets for the selected blocks and fields automatically. The system can run everything at one time. It can also be run separately depending on the need. In the SAS code development part, the system can produce field lists, output control lists, and SAS code separately. Users have the option to create the ASCII relational data sets and the SAS data sets at different times. In the data export system, we also include a program to create the timing information for each section in the CE data model. It reads time information for each field in the audit trail file and computes the total time for each section and creates a SAS data set.

The output control file plays the key role in the data export system. It drives the entire output processing. The advantages of this system are:

- **Flexibility**

It can output any number of SAS data sets depending on how the output control file is created. For example, if subject matter experts just want to review one data set, they can create an output control file for the data set. It can also output any number of the fields. It is especially useful during the early stages of data output development and testing.

- **Efficiency**

The system can export each stage's data separately. It avoids repeating the same output procedure if the change in the output control file does not effect the early stages of data output. It is very useful for large data sets. Most of our data processing is run on a Unix system during production. The data export system allows us to perform our tasks on the Unix system once the ASCII relational data sets are created in the microcomputer environment.

- **Easy to maintain and adapt to other surveys**

The data export system is independent of any specific Blaise data model. The data model and output control files are the inputs to the system. The system itself does not need to be modified if there are changes in these inputs. The output can be obtained by re-running the system with a corrected output control file and data model. The system can be used to export data for other Blaise data models. The user would just have to identify the data model name and the control file name.

The goal of the data export system is to output data from the Blaise data model and to generate SAS data sets for selected blocks and fields automatically. We have discussed all the steps needed to achieve this goal. The data export system for the CE data model is based on three assumptions:

- The block structure of the CE data model corresponds well to the requirements of output data sets.
- The data model is well defined and organized.
- The output control file is easy to update and can drive the data export system.

The data export system for the CE data model can not be successful without these assumptions. The export system could not have been built without the work of the specification writers, data model developers, and subject matter experts.

References

1. Statistics Netherlands (1999) Blaise Developer's Guide
2. Pierzhchala, Mark and Farrant , Graham (2000) "Helping non-Blaise Programmers to Specify a Blaise Instrument", Proceedings of the 6th International Blaise Users Conference
3. Frey, Richard (2000) "Developing Blaise instrument for the Spanish Bladder Cancer Survey", Proceedings of the 6th International Blaise Users Conference

Session 3: Case Management I

- LAURA
Marko Sluga and Janez Repinc
Statistical Office of the Republic of Slovenia
- Interactive Shelling Between Visual Basic and Blaise
John Cashwell, Battelle, USA
- Standardised Survey Management at Statistics Netherlands: An Example of the Object Approach
Frans Kerssemakers, Marien Lina, and Lon Hofman
Statistics Netherlands`
- Survey of Labour and Income Dynamic (SLID) Project at Statistics Canada
Richard Chan, Statistics Canada

LAURA (Lahka Administracija Uporabnikov, Raziskav in Aplikacij) Easy Administration of Users, Surveys and Applications

Janez Repinc, Statistical Office Of The Republic Of Slovenia
Marko Sluga, Statistical Office Of The Republic Of Slovenia

1. Introduction

Laura is system for administration of users and statistical surveys developed with Blaise. We got the first idea at the end of 1999, development started in the middle of 2000. Main reason for development was insufficient controls over interviewers work, their user rights, cumbersome administration of surveys and dynamic change of environment variable BlaiseUser in WinNT 4 operating system. Before all this problems were solved with unique applications developed with Microsoft VisualBasic, but with time and growing number of different surveys this was not long time solution.

2. Overview

Core of Laura system is Microsoft Access database where all data about users, surveys, user levels and their relationships is stored. With this data Laura can generate different user interfaces which are more user friendly because of their standardised looks and availability of only those functions that can be run on a certain user level. It also drastically decreases needed developers work because there is no need for several different user interfaces. If functionality of generated user interface is not good enough Laura can generate initialisation file with all relevant parameters, so it can be used with any other developed application. For use of this file to be as easy as possible we developed ActiveX DLL library with all needed functions. With use of WinVNC programme Laura enables remote control of all interviewer computers.

3. Database

Laura's database is developed in MS Access. It contains all information needed for system to work. The data is organised in seven main categories:

- user informations: user name, password, real name, default user level, BlaiseUser
- survey information: survey code, short name, long name, programmer ID, path to survey folder, Blaise version, parameters for generated user interface (font, color,...), optional path to external user interface
- user levels information
- informations of relations between users and surveys
- Blaise informations: all paths to Blaise programs
- log informations: all activities are logged
- informations needed by system

4. User groups

One of the main features in Laura are user groups because they can be used to activate advanced features in user interfaces. Laura differentiates between five user groups:

- interviewer: can run surveys which were assigned to him
- controller: interviewer rights + can add new users (interviewer level only), can assign users to surveys, can overview interviewers work with help of short statistics and remote control
- supervisor: controller rights + can archive data, can assign controller rights
- programmer: supervisor rights + can add/edit survey which he developed
- administrator: no restrictions

5. User Interface

User Interface is developed with MS Visual Basic, standard SQL language is used for accessing database, more advanced features use WinAPI functions, for data crypting we used DES3 algorithm.

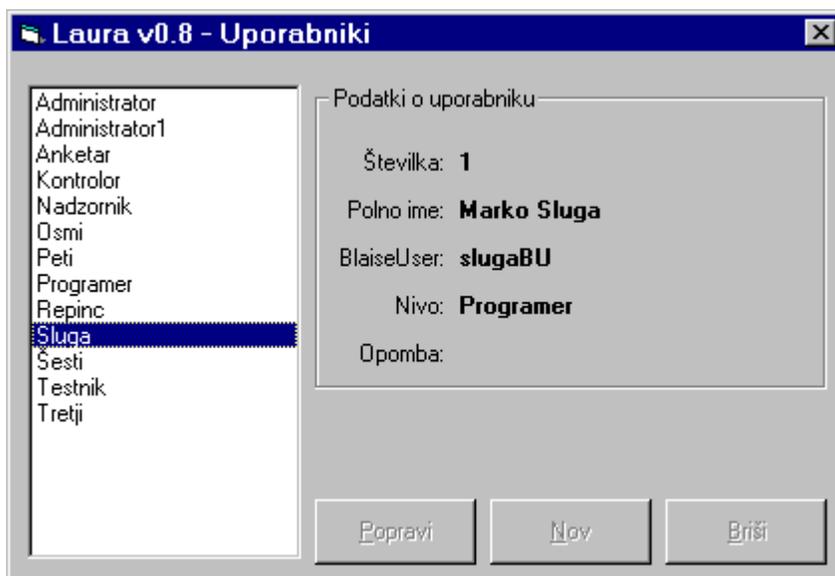
After starting Laura must every user login to system with user name and password. If login is successful Lauras main window opens and user data with start time is written to log. If user is in the administrators group then he gets the list of all surveys and all functions are enabled, otherwise only assigned surveys are. Main window contains list box with short survey names assigned to current user sorted alphabetically. Upon selecting survey more details are shown and based on survey user group equivalent advanced functions are enabled.

6. Administration

With buttons on main window you can administer users, surveys and their relations. These buttons are enabled only when current user is in specified group, so controller and supervisor can administer users, programmer can add and edit surveys wich he developed. Users from administrators group don't have any restrictions.

6.1. User Management

When we start user management we get listbox with all usernames sorted alphabetically and buttons for adding, editing and deleting users. If we want to add new user, after clicking on button 'New', we get new window where we fill in specified fields. Default user group higher than interviewer can be set only by administrator. No one can edit user in higher or same user group unless they are in administrators group. The 'Delete' button in enabled only to administrators.



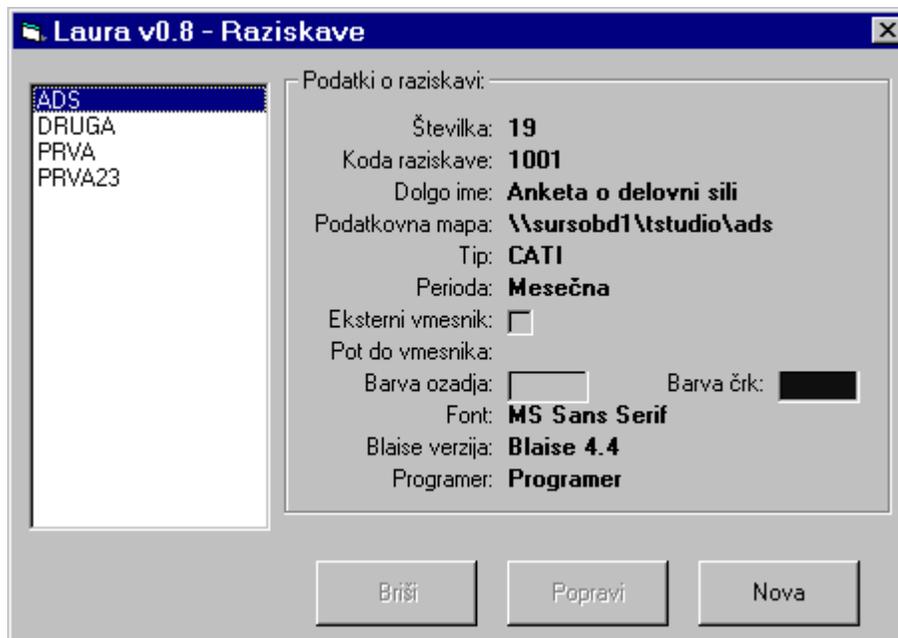
Picture 1: Main user management screen with user details and buttons for adding, editing or deletin users



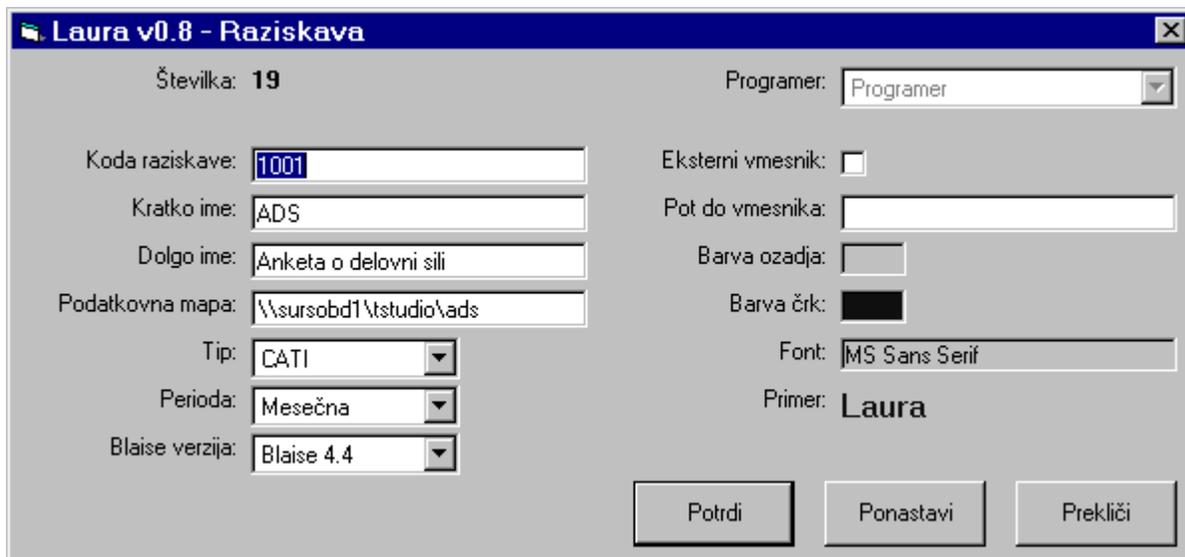
Picture 2: User management edit screen

6.2. Survey Management

Survey management is available only to programmers and administrators. In main survey management window we get listbox with short survey names and buttons for adding, editing and deleting surveys. Each programmer can edit and delete only those surveys that he developed. If we want to add new survey we can click on 'New' button and fill in all fields.



Picture 3: Main survey management screen with survey details and buttons for adding, editing or deleting users

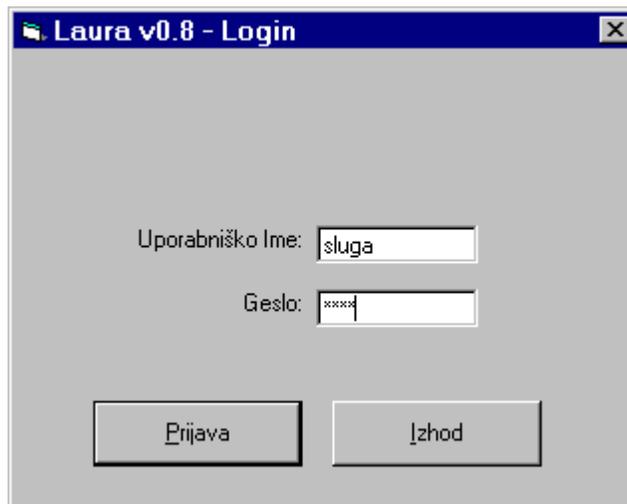


Picture 4: Survey management edit screen.

6.3. User-Survey Management

The button for user-survey management on main window is enabled to all users from controller group and higher. First assigning of users to survey can only be done by programmer or administrators. If we want to assign new users or change their survey user group, we must select survey from listbox and after clicking 'User-Survey' button we get new window with four listboxes (one for all users and other three for users in different survey user groups). We can add or remove user simply by double-clicking his username. When we add user to survey his survey user group is the same as default user group. We can move user from one group to another with buttons on the interface.

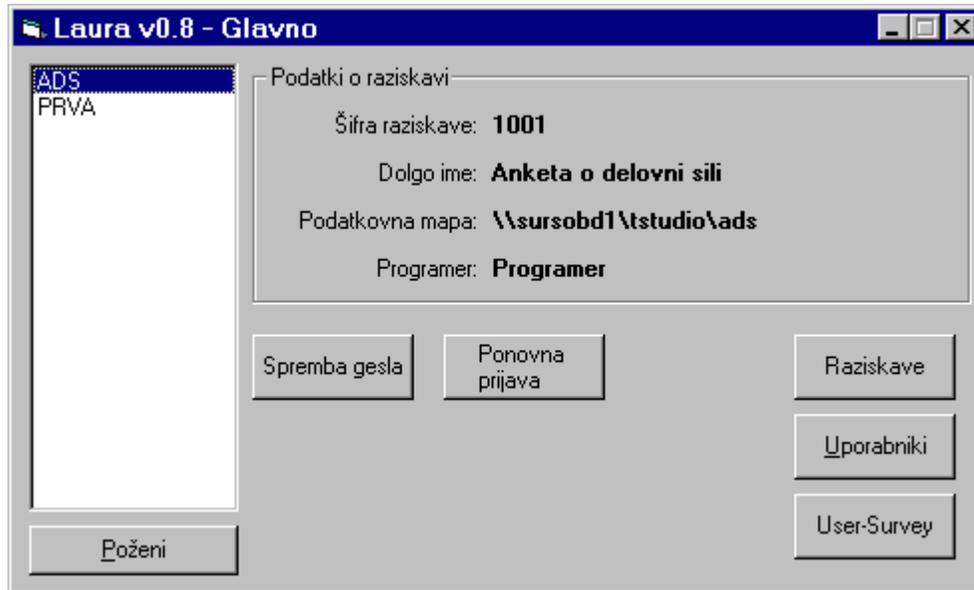
7. Working with Laura



Picture 5: Login screen.

With double-click or click on 'Start' button we run selected survey. Depending on parameters in survey definition we get on screen generated user interface for that survey or external user interface. If current user is in interviewer group, he gets only buttons for running DEP and help. Controller has added buttons

for daily backup, statistics, remote control and if survey is in CATI mode also buttons for history and CATI management. User interface for supervisor or higher has two additional buttons; for archiving and CATI specification. If survey uses external user interface, Laura generates initialization file with all needed user and survey data.



Picture 6: Main screen with all buttons for administration enabled.

8. Overviewing work

8.1. Statistics

All user activity is logged. For every survey can Laura show data for active users such as username, Windows login, computer name, computer IP, ethernet address, start time (when Laura was started), survey name, session time (when survey was started).

8.2. Remote control

Instead of developing our own application for remote control we decided to use WinVNC developed by AT&T Labs Cambridge which is available freely and with source code. For our needs the code was modified so that interviewers don't have any control over WinVNC server. We can run WinVNC client by clicking on user in statistics window. The client connects to specified server on user computer and new window opens with screen in real time.

9. Security Functions

Security features can be organized in three categories:

- system security: the database is password protected; all important data is crypted; also the application is crypted
- activity logging: data for every logged user is written to log table including data about computer from which login was successful (windows login, ethernet address, computer IP) and start and end date and time. All remote control use is also logged to separate table with data about users and computers on both server and client side with start time.
- remote control security: to prevent possible misuse of remote control WinVNC server is run only when Laura starts. The password for remote control is changed dynamically and crypted with DES3 algorithm. When WinVNC client connects to server the use of local mouse and keyboard is disabled.

10. Conclusions

Because of constant development and implementing of new functions Laura is still in test phase and as such is not suitable for production use. We hope that first production version will be available in early 2002.

Interactive Shelling Between Visual Basic And Blaise

John P. Cashwell, Battelle

The Blaise survey processing system is an effective solution for collecting survey data, however complicated studies may require more sophisticated project management and tracking functionality. The required functionality may be offered by a Visual Basic application. This application could operate as a shell around a Blaise computer-assisted instrument (CAI), helping users to comply with project timelines and parameters.

One of the challenges in developing such a system is creating a streamlined method for shelling from Visual Basic to a Blaise survey. There are many alternatives, however some are not compatible across operating systems, particularly between Windows NT/2000 and Windows 98/95. While some solutions may function correctly, the user interface may be awkward or unappealing. This paper will review some of the less desirable methods and conclude with a recommended solution.

First, it is worthwhile to quickly review some less desirable shelling methods. These methods usually work, however each one has a downside. The less desirable solutions covered in this paper will involve DOS batch files and Windows API calls. Sample Visual Basic code for these solutions will be reviewed.

After reviewing some less desirable methods, this paper will reveal a simple and effective solution that does not have the problems explained in the previous section. Sample Visual Basic code for this solution will be reviewed. The Visual Basic code will first call a Manipula program to pass data from a Microsoft Access database into a Blaise database. That same code will then run a Blaise Computer Assisted Interview (CAI). Upon terminating the CAI, Visual Basic will then call a Manipula program to determine the CAI completion status (e.g. complete, partial complete, etc.).

This solution will not require DOS batch files or Windows API calls and all calls to Manipula.exe and Dep.exe will be made from within the Visual Basic code. In addition, the code will ensure that all processing occurs one request at a time in order (synchronously) and will be compatible between the Windows NT/2000 and Windows 98/95 operating systems.

Less Desirable Shelling Methods

DOS Batch Files

DOS Batch files may be used to synchronously process several requests. Calling processes **synchronously** will ensure that each process will not start until the preceding process is finished. For example, you might need to run a manipula program to add or update a Blaise record before running the datamodel. Then you might want to export data from that same datamodel after closing it. Here is a sample DOS batch file that illustrates the above.

```
rem THIS IS A SAMPLE BATCH FILE THAT ADDS A RECORD TO
rem A BLAISE DATAMODEL VIA addid.man THEN RUNS THE DATAMODEL (cati_1).
rem AFTER THE DATAMODEL IS CLOSED, IT RUNS result.man TO
rem EXPORT RESULTS FROM THE DATAMODEL TO AN ASCII FILE.
@echo off
cls
echo Initializing interview.....

rem SET WORKING DRIVE
K:

rem SET WORKING DIRECTORY
cd\projects\cati

rem 'Start /w' IS REQUIRED WHEN NEEDING TO RUN WINDOWS PROGRAMS SYNCHRONOUSLY.
rem WITHOUT START /W, EACH CALL TO A WINDOWS PROGRAM WILL RUN AT THE SAME TIME INSTEAD
rem INSTEAD OF IN ORDER.
rem 'Start /w' IS NOT NEEDED TO RUN DOS PROGRAMS SYNCHRONOUSLY.

rem RUN THE MANIPULA PROGRAM addid.man TO ADD/UPDATE A RECORD.
rem '%1' IS A PARAMETER SENT INTO THE BATCH FILE AND THEN SENT INTO MANIPULA.
Start /w k:\blaise\windows\ver_452b642\manipula.exe addid /q /p%1

rem RUN THE DATAMODEL cati_1 AFTER THE addid.man IS FINISHED.
rem '%1' IS A PARAMETER SENT INTO THE BATCH FILE AND THEN SENT INTO DEP.
Start /w k:\blaise\windows\ver_452b642\dep.exe cati_1 /g /k%1 /x

rem RUN THE MANIPULA PROGRAM result.man TO GET CERTAIN RESULTS EXPORTED FROM
rem THE cati_1 DATAMODEL AFTER EXITING dep.exe.
rem '%1' IS A PARAMETER SENT INTO THE BATCH FILE AND THEN SENT INTO MANIPULA.
Start /w k:\blaise\windows\ver_452b642\manipula.exe result /q /p%1

rem NOTE THAT WE USE Start /w ON THE LAST CALL TOO. THIS DOES NOT MAKE SENSE
rem IF YOU ARE ONLY INTERESTED IN MAKING THE CALLS IN THE DOS BATCH FILE
rem SYNCHRONOUS. HOWEVER, IF YOU WANT TO ALSO SYNCHRONOUSLY CALL THE DOS
rem BATCH FILE WITHIN ANOTHER PROGRAM (I.E. VISUAL BASIC), YOU WANT THE
rem DOS BATCH FILE TO REMAIN OPEN UNTIL ALL PROCESSES INSIDE IT ARE FINISHED.

rem CLOSE THE BATCH FILE WINDOW
Exit
```

The downsides of this approach are that it creates another link in the programming chain that further complicates the programmer's job, DOS Batch files may not be supported in the future, and DOS may pop-up in a window that confuses the user. It also raises code security concerns unless the programmer changes the properties of the DOS batch file to prohibit users from editing or deleting it.

You may call a DOS batch file within a Visual Basic project with the Shell command.

Example:

```
Dim RetVal
RetVal = Shell("C:\Sample\Sample.bat", 1)
```

According to Microsoft, *“By default, the Shell function runs other programs **asynchronously**. This means that a program started with Shell might not finish executing before the statements following the*

Shell function are executed.” This restricts your control within Visual Basic. Other means are necessary if you need to run several commands synchronously. The following Windows API methods can solve that problem. However an even better solution will be covered later.

Windows API Functions

In a Visual Basic Project, you might have code like the following examples to synchronously shell to external programs. This code incorporates Windows API functions. On first glance this code is complicated and not for the novice programmer.

```
Option Explicit

Public Const GW_HWNDFIRST = 0
Public Const GW_HWNDLAST = 1
Public Const GW_HWNDNEXT = 2
Public Const GW_HWNDPREV = 3

Public Const GW_CHILD = 5
Public Const GW_MAX = 5

Global Const NORMAL_PRIORITY_CLASS = &H20&
Global Const INFINITE = -1&

Public Const STILL_ACTIVE = &H103
Public Const PROCESS_QUERY_INFORMATION = &H400

Public Const GW_OWNER = 4
Public Const GWL_STYLE = -16
Public Const WS_DISABLED = &H8000000
Public Const WS_CANCELMODE = &H1F
Public Const WM_CLOSE = &H10

Public glCurrentHwnd

Type STARTUPINFO
    cb As Long
    lpReserved As String
    lpDesktop As String
    lpTitle As String
    dwX As Long
    dwY As Long
    dwXSize As Long
    dwYSize As Long
    dwXCountChars As Long
    dwYCountChars As Long
    dwFillAttribute As Long
    dwFlags As Long
    wShowWindow As Integer
    cbReserved2 As Integer
    lpReserved2 As Long
    hStdInput As Long
    hStdOutput As Long
    hStdError As Long
End Type

Type PROCESS_INFORMATION
    hProcess As Long
    hThread As Long
    dwProcessID As Long
    dwThreadID As Long
End Type

'THESE DECLARE FUNCTIONS ARE ALL WINDOWS API CALLS
'CONFUSING AND HARD TO USE FOR NOVICE PROGRAMMERS.
'ALSO REQUIRES ADDITIONAL SYSTEM RESOURCES
Declare Function OpenProcess Lib "kernel32" _
    (ByVal dwDesiredAccess As Long, ByVal bInheritHandle _
    As Long, ByVal dwProcessID As Long) As Long
Declare Function GetExitCodeProcess Lib "kernel32" _
```

```

    (ByVal hProcess As Long, lpExitCode As Long) As Long
Declare Function CloseHandle Lib "kernel32" (hObject As Long) _
    As Boolean
Declare Function WaitForSingleObject Lib "kernel32" (ByVal _
    hHandle As Long, ByVal dwMilliseconds As Long) As Long
Declare Function CreateProcessA Lib "kernel32" _
    (ByVal lpApplicationName As Long, ByVal lpCommandLine As _
    String, ByVal lpProcessAttributes As Long, ByVal _
    lpThreadAttributes As Long, ByVal bInheritHandles As _
    Long, ByVal dwCreationFlags As Long, ByVal _
    lpEnvironment As Long, ByVal _
    lpCurrentDirectory As Long, lpStartupInfo As STARTUPINFO, _
    lpProcessInformation As PROCESS_INFORMATION) As Long
Declare Function IsWindow Lib "user32" (ByVal hwnd As Long) _
    As Long
Declare Function GetWindow Lib "user32" (ByVal hwnd As Long, _
    ByVal wCmd As Long) As Long
Declare Function PostMessage Lib "user32" Alias _
    "PostMessageA" (ByVal hwnd As Long, ByVal wParam _
    As Long, ByVal lParam As Long) As Long
Declare Function GetWindowLong Lib "user32" Alias _
    "GetWindowLongA" (ByVal hwnd As Long, _
    ByVal nIndex As Long) As Long
Declare Function GetParent Lib "user32" (ByVal hwnd _
    As Long) As Long
Declare Function GetWindowTextLength Lib "user32" Alias _
    "GetWindowTextLengthA" (ByVal hwnd As Long) As Long
Declare Function GetWindowText Lib "user32" Alias _
    "GetWindowTextA" (ByVal hwnd As Long, ByVal lpString _
    As String, ByVal cch As Long) As Long

Dim strAppToLaunch As String

`USE SHELLANDCONTINUE TO ALLOW THE VISUAL BASIC
`TRACKING SYSTEM TO STILL BE USED WHILE RUNNING THE SPECIFIED PROCESS.
`THIS PROCESS STILL KEEPS TRACK OF WHEN THE CALLED PROCESS IS
`DONE PROCESSING. WHEN THAT HAPPENS, THE CODE CONTINUES WITHIN
`THE SHELLANDCONTINUE FUNCTION. SO, IT REALLY DOES STOP EXECUTION
`WITHIN THAT FUNCTION, BUT STILL ALLOWS USER TO DO OTHER THINGS IN THE
`TRACKING SYSTEM.

Call ShellAndContinue(strAppToLaunch)

`OR YOU CAN USE SHELLANDWAIT TO CAUSE THE VISUAL BASIC TRACKING SYSTEM
`TO COMPLETELY LOCKUP WHILE RUNNING THE SPECIFIED PROCESS.
`THIS PROHIBITS THE USER FROM DOING ANYTHING IN THE TRACKING SYSTEM
`WHILE THE PROCESS IS STILL RUNNING. THIS IS CUMBERSOME AND AT TIMES
`CONFUSING TO THE END USER, BUT MAKES LIFE EASIER FOR THE PROGRAMMER
`BECAUSE THEY DON'T HAVE TO WORRY ABOUT CONTROLLING USER EVENTS WHILE
`RUNNING THE EXTERNAL PROCESS.

Call ShellAndWait(strAppToLaunch)

....

Sub ShellAndContinue(ByVal AppToRun As String)
    On Error GoTo ErrorRoutineErr

    Dim hProcess As Long
    Dim RetVal As Long
    Dim Msg, Style, Title, Response 'msgbox variables

    'The next line launches AppToRun,
    'captures process ID
    hProcess = OpenProcess(PROCESS_QUERY_INFORMATION, 1, _
    Shell(AppToRun, vbNormalFocus))

```

```

Do
    'Get the status of the process
    GetExitCodeProcess hProcess, RetVal
    DoEvents
'Loop while the process is active
Loop While RetVal = STILL_ACTIVE

'define message
Msg = AppToRun & " Terminated by user"
'define buttons
Style = vbOKOnly + vbInformation
'define title
Title = "Termination Notice"
'display message
Response = MsgBox(Msg, Style, Title)

ErrorRoutineResume:
Exit Sub
ErrorRoutineErr:
MsgBox "AppShell.Form1.ShellAndContinue" & Err & Error
End Sub

Public Sub ShellAndWait(AppToRun)
On Error GoTo ErrorRoutineErr

Dim NameOfProc As PROCESS_INFORMATION
Dim NameStart As STARTUPINFO
Dim rc As Long

NameStart.cb = Len(NameStart)
rc = CreateProcessA(0&, AppToRun, 0&, 0&, 1&, _
    NORMAL_PRIORITY_CLASS, _
    0&, 0&, NameStart, NameOfProc)
rc = WaitForSingleObject(NameOfProc.hProcess, INFINITE)
rc = CloseHandle(NameOfProc.hProcess)

ErrorRoutineResume:
Exit Sub
ErrorRoutineErr:
MsgBox "AppShell.Form1.ShellAndWait" & Err & Error
Resume Next

End Sub

```

The problems with the above code are that it is confusing and resource intensive. Also, some of it cannot be used to call MANIPULA.EXE or DEP.EXE. Specifically, it seems that the ShellAndWait method completely locks up the application when attempting to call MANIPULA or DEP. This happens when the WaitForSingleObject function is called. This problem forces the programmer to wrap up the MANIPULA and DEP calls inside a DOS batch file or some other kind of scripting file and then call that script file with the above procedures. This starts to be an entangled mess that is hard to debug and confusing for programmers. In case you are wondering, the ShellAndWait function works for other applications like NOTEPAD.EXE. The problem seems to be Blaise related.

RECOMMENDED METHOD FOR SYNCHRONOUS SHELLING

There are other ways to synchronously shell from Visual Basic to MANIPULA and/or DEP, however we should move on to this paper's recommended way to do the same thing. The heart of the recommended code replaces all of the code shown above. This code is bolded inside the class 'clsProcess'. It's amazing how brief this code is compared to the ShellAndWait and ShellAndContinue examples.

Here is the code that replaces the code above:

```
'FOR STORING THE MANIPULA/DEP COMMAND STRING
Dim strCommand as String
'FOR CONTROLLED SHELLING TO MANIPULA/DEP
Dim WshShell As IWshShell

'CREATE THE WSHSHELL OBJECT
Set WshShell = CreateObject("Wscript.Shell")

'CREATE THE VARIABLE TO RECEIVE THE RETURN VALUE FROM WSHSHELL
Dim lngAppReturnValue As Long

'RUN THE MANIPULA/DEP COMMAND STRING, TELLING IT TO USE A MAXIMIZED WINDOW (3)
'AND SET THE BOOLEAN PARAMETER TO 'TRUE' SO WSHSHELL WILL WAIT UNTIL THE PROCESS IS FINISHED
'BEFORE PROCEEDING TO THE NEXT LINE OF CODE.
lngAppReturnValue = WshShell.Run(strCommand, 3, True)
```

Microsoft's Windows Script Host (WshShell) object exposes some of the common shell functionalities of Microsoft Windows, making it easy to create shortcuts, access the environment variables, run applications, access system registry, and more. The WshShell object is not instantiated automatically upon script execution, hence it must be instantiated explicitly using CreateObject before it can be used.

The following code is organized into Class modules. A programmer can instantiate the code as an object and send the required parameters to the appropriate methods. This frees the programmer from having to repeatedly delve into more complicated code. It also allows for easier upgrade of code, because the code is only repeated once. Specifically, these classes shield the front-end Visual Basic programmer who doesn't know much about Blaise from having to get into the details of how to call a Manipula program or datamodel.

It's important to note that the classes specified below are not required. The additional code was added to make life easier for the 'front-end' programmers who design the end user's Visual Basic forms. If you are not interested in using these classes, you can simply use the few lines of code bolded at the beginning of this section.

```

'Programmer Notes:
'Microsoft® Windows® Script Host
'ACCORDING TO MICROSOFT:
'object.Run(strCommand, [intWindowState], [bWaitOnReturn])
'STRCOMMAND =
' STRING VALUE INDICATING THE COMMAND LINE YOU WANT TO RUN.
' YOU MUST INCLUDE ANY PARAMETERS YOU WANT TO PASS TO THE EXECUTABLE FILE.
'INTWINDOWSTYLE (OPTIONAL) =
' INTEGER VALUE INDICATING THE APPEARANCE OF THE PROGRAM'S WINDOW. NOTE THAT NOT ALL
' PROGRAMS MAKE USE OF THIS INFORMATION.
'BWAITONRETURN (OPTIONAL) =
' BOOLEAN VALUE INDICATING WHETHER THE SCRIPT SHOULD WAIT FOR THE PROGRAM TO FINISH
' EXECUTING BEFORE CONTINUING TO THE NEXT STATEMENT IN YOUR SCRIPT. IF SET TO TRUE,
' SCRIPT EXECUTION HALTS UNTIL THE PROGRAM FINISHES, AND RUN RETURNS ANY ERROR CODE
' RETURNED BY THE PROGRAM (0 FOR SUCCESS). IF SET TO FALSE (THE DEFAULT), THE RUN METHOD
' RETURNS IMMEDIATELY AFTER STARTING THE PROGRAM, AUTOMATICALLY RETURNING 0 (NOT TO BE
' INTERPRETED AS AN ERROR CODE).
'
'REMARKS
' THE RUN METHOD RETURNS AN INTEGER. THE RUN METHOD STARTS A PROGRAM RUNNING IN A NEW
' WINDOWS PROCESS. YOU CAN HAVE YOUR SCRIPT WAIT FOR THE PROGRAM TO FINISH EXECUTION
' BEFORE CONTINUING. THIS ALLOWS YOU TO RUN SCRIPTS AND PROGRAMS SYNCHRONOUSLY.
' ENVIRONMENT VARIABLES WITHIN THE ARGUMENT STRCOMMAND ARE AUTOMATICALLY EXPANDED.
' IF A FILE TYPE HAS BEEN PROPERLY REGISTERED TO A PARTICULAR PROGRAM, CALLING RUN ON A
' FILE OF THAT TYPE EXECUTES THE PROGRAM. FOR EXAMPLE, IF WORD IS INSTALLED ON YOUR
' COMPUTER SYSTEM, CALLING RUN ON A *.DOC FILE STARTS WORD AND LOADS THE DOCUMENT.
' THE FOLLOWING TABLE LISTS THE AVAILABLE SETTINGS FOR INTWINDOWSTYLE.
'
'INTWINDOWSTYLE DESCRIPTION
'0 HIDES THE WINDOW AND ACTIVATES ANOTHER WINDOW.
'1 ACTIVATES AND DISPLAYS A WINDOW. IF THE WINDOW IS MINIMIZED OR MAXIMIZED,
' THE SYSTEM RESTORES IT TO ITS ORIGINAL SIZE AND POSITION. AN APPLICATION
' SHOULD SPECIFY THIS FLAG WHEN DISPLAYING THE WINDOW FOR THE FIRST TIME.
'2 ACTIVATES THE WINDOW AND DISPLAYS IT AS A MINIMIZED WINDOW.
'3 ACTIVATES THE WINDOW AND DISPLAYS IT AS A MAXIMIZED WINDOW.
'4 DISPLAYS A WINDOW IN ITS MOST RECENT SIZE AND POSITION. THE ACTIVE WINDOW REMAINS ACTIVE.
'5 ACTIVATES THE WINDOW AND DISPLAYS IT IN ITS CURRENT SIZE AND POSITION.
'6 MINIMIZES THE SPECIFIED WINDOW AND ACTIVATES THE NEXT TOP-LEVEL WINDOW IN THE Z ORDER.
'7 DISPLAYS THE WINDOW AS A MINIMIZED WINDOW. THE ACTIVE WINDOW REMAINS ACTIVE.
'8 DISPLAYS THE WINDOW IN ITS CURRENT STATE. THE ACTIVE WINDOW REMAINS ACTIVE.
'9 ACTIVATES AND DISPLAYS THE WINDOW. IF THE WINDOW IS MINIMIZED OR MAXIMIZED,
' THE SYSTEM RESTORES IT TO ITS ORIGINAL SIZE AND POSITION. AN APPLICATION SHOULD
' SPECIFY THIS FLAG WHEN RESTORING A MINIMIZED WINDOW.
'10 SETS THE SHOW-STATE BASED ON THE STATE OF THE PROGRAM THAT STARTED THE APPLICATION.

```

```

'*****
'CLASS: clsInterview
'PURPOSE: Allow easy front end management of external Blaise interviews
'*****
Option Explicit

'USER DEFINED OBJECT FOR RUNNING APPLICATION PROCESSES
'OUTSIDE OF THE VISUAL BASIC APPLICATION
Private moProcess As New clsProcess
'String HOLDING LOCATION OF THE MANIPULA.EXE TO USE
Const MANIPULA_EXE = "K:\PROJECTS\CATI\BLAISE\WINDOWS\Ver_424B518\MANIPULA.EXE"
'String HOLDING LOCATION OF THE DEP.EXE TO USE
Const DEP_EXE = "K:\PROJECTS\CATI\BLAISE\WINDOWS\Ver_424B518\DEP.EXE"

'*****
'PURPOSE: TO START A SPECIFIED INTERVIEW NUMBER
'INPUTS: INTERVIEW #, SUBJECT ID, DIRECTORY OF INPUT FILES (OPTIONAL),
'        DATA STRING BEING PASSED TO INPUT FILE (OPTIONAL)
'RETURNS: BOOLEAN TRUE MEANS SUCCESSFULLY STARTED INTERVIEW
'        BOOLEAN FALSE MEANS ATTEMPT TO START INTERVIEW FAILED
'PROGRAMMER: JOHN CASHWELL
'LAST UPDATED: 7.31.2001
'*****
Public Function StartInterview(intInterviewNumber As Integer, strID As String, Optional
strInputDirectory As String, Optional strData As String) As Boolean
On Error GoTo ERR_HANDLER

    StartInterview = False

    Select Case intInterviewNumber
    Case 1
        'WRITE THE PARAMETER strData TO AN ASCII FILE WHICH WILL BE IMPORTED INTO BLAISE
        Open strInputDirectory & "\" & Trim$(strID) & ".in" For Output As #1
        Print #1, strData
        Close #1

        'RUN MANIPULA TO INITIALIZE CASE IN BLAISE
        If moProcess.RunManipulaProcess( _
            MANIPULA_EXE, _
            "START.MAN", _
            "K:\PROJECTS\CATI\CPHRE_TRACKING\BLAISE\MAN", _
            "K:\PROJECTS\CATI\CPHRE_TRACKING\BLAISE", _
            True, False, _
            "K:\PROJECTS\CATI\CPHRE_TRACKING\BLAISE\INPUT\" & Trim$(strID) & ".in" _
            , , Trim$(strID)) = False Then
            Exit Function
        End If
        'RUN CATI INTERVIEW
        If moProcess.RunDEPProcess(DEP_EXE, "K:\PROJECTS\CATI\CPHRE_TRACKING\BLAISE", _
            "INTERVIEW", strID, True, " ") = False Then
            Exit Function
        End If

        'RUN MANIPULA TO RETURN RESULTS
        If moProcess.RunManipulaProcess( _
            MANIPULA_EXE, "END.MAN", _
            "K:\PROJECTS\CATI\CPHRE_TRACKING\BLAISE\MAN", _
            "K:\PROJECTS\CATI\CPHRE_TRACKING\BLAISE", _
            True, False, , _
            "K:\PROJECTS\CATI\CPHRE_TRACKING\BLAISE\OUTPUT\" & Trim$(strID) & ".OUT", _
            Trim$(strID)) = False Then
            Exit Function
        End If
        'SUCCESSFUL COMPLETION OF ALL TASKS
        StartInterview = True
    Case 2
        'RESERVED FOR FOLLOW-UP INTERVIEWS

```

```

        'CASE 3 ETC..
    End Select

    Exit Function
ERR_HANDLER:
    Call ErrorHandler
End Function

'*****
'PURPOSE:  TO READ INTERVIEW RESULT FROM A SPECIFIED FILE
'INPUTS:   DIRECTORY OF INPUT FILE, NAME OF INPUT FILE
'RETURNS:  RESULT STRING (i.e. C=COMPLETED, P=PARTIAL, etc..)
'PROGRAMMER:  JOHN CASHWELL
'LAST UPDATED:  7.31.2001
'*****
Public Function GetResult(strDirectory As String, strFileName As String) As String
On Error GoTo ERR_HANDLER
    Dim strRow As String

    If Dir(strDirectory & "\" & strFileName) = "" Then
        MsgBox "The result file '" & strDirectory & "\" & strFileName & "' " & _
            "does not exist. Please notify programmer."
        Exit Function
    End If

    Open strDirectory & "\" & strFileName For Input As #1

    Do Until EOF(1)
        Line Input #1, strRow
    Loop

    GetResult = strRow

    GoTo CLEAN_UP
ERR_HANDLER:
    Call ErrorHandler
    GoTo CLEAN_UP
CLEAN_UP:
    Close #1
End Function

```

```

'*****
'CLASS: clsProcess
'PURPOSE: To control Manipula and DEP processes.
'*****
Option Explicit

'*****
'PURPOSE: TO EASILY RUN MANIPULA PROGRAMS AND CONTROL THE ORDER
' OF EVENTS BETWEEN THE TRACKING SYSTEM AND MANIPULA
'INPUTS: MANIPULA.EXE'S PATH AND FILE NAME,
'        MANIPULA PROGRAM'S NAME,
'        MANIPULA PROGRAM'S DIRECTORY PATH,
'        DATAMODEL'S DIRECTORY PATH,
'        BOOLEAN FOR WHETHER TO RUN MANIPULA IN QUIET MODE,
'        BOOLEAN FOR WHETHER TO VIEW MANIPULA RESULTS WHEN DONE PROCESSING
'        (Quite Mode Must = False To Have View Results = True),
'        MANIPULA INPUT FILE PATH AND FILE NAME (OPTIONAL),
'        MANIPULA OUTPUT FILE PATH AND FILE NAME (OPTIONAL),
'        PARAMETERS TO SEND INTO MANIPULA PROGRAM (OPTIONAL),
'        ANY ADDITIONAL OPTIONS (OPTIONAL)
'RETURNS: BOOLEAN TRUE IF SUCCESSFULLY RUNS MANIPULA PROCESS
'PROGRAMMER: JOHN CASHWELL
'LAST UPDATED: 7.31.2001
'*****
Public Function RunManipulaProcess( _
    strManipula_EXE_Path_And_File_Name, _
    strManipula_Program_Name As String, _
    strManipula_Program_Path As String, _
    strDatamodel_Path As String, _
    blnQuiet As Boolean, _
    blnStop_And_View_Results As Boolean, _
    Optional strInput_File_Path_And_Name As String, _
    Optional strOutput_File_Path_And_Name As String, _
    Optional strParameters As String, _
    Optional strAdditional_Options As String) As Boolean
On Error GoTo ERR_HANDLER

'IN ORDER TO USE THE IWSHSHELL OBJECT, YOU NEED TO HAVE
'A REFERENCE IN VISUAL BASIC TO: WSHOM.OCX
'IT IS CALLED 'WINDOWS SCRIPT HOST OBJECT MODEL'

Dim WshShell As IWshShell 'FOR CONTROLLED SHELLING TO MANIPULA
Dim strDriveLetter As String 'TO SET WORKING DRIVE
Dim strCommand As String 'TO BUILD THE FULL MANIPULA COMMAND STRING

'GET DRIVER LETTER
strDriveLetter = Left(strManipula_Program_Path, 1)

'INITIALIZE RETURN VALUE TO FALSE
RunManipulaProcess = False

'CREATE THE WSHSHELL OBJECT AS WINDOWS HOST SCRIPT OBJECT
'THIS OBJECT IS USED TO CONTROL SHELLING TO EXTERNAL
'APPLICATION LIKE BLAISE
Set WshShell = CreateObject("Wscript.Shell")

'CHANGE DIRECTORY TO WHERE THE MANIPULA PROGRAM RESIDES
If Dir(strManipula_Program_Path, vbDirectory) = "" Then
    MsgBox "The directory specified for " & strManipula_Program_Name & _
        " does not exist. Process cancelled."
    Exit Function
End If

'CHANGE DRIVE AND DIRECTORY TO FOLDER THE MANIPULA PROGRAMS RESIDE
ChDrive strDriveLetter
'CHANGE THE DIRECTORY
ChDir strManipula_Program_Path

```

```

'BUILD THE COMMAND STRING FROM THE OPTIONS PASSED IN TO FUNCTION
strCommand = strManipula_EXE_Path_And_File_Name
strCommand = strCommand & " " & strManipula_Program_Name
strCommand = strCommand & " /F" & strDatamodel_Path
strCommand = strCommand & IIf(blnQuiet, " /Q", "")
strCommand = strCommand & IIf(blnStop_And_View_Results, " /A", "")
strCommand = strCommand & IIf(Trim$(strInput_File_Path_And_Name) = "", "", " /I" &
    strInput_File_Path_And_Name)
strCommand = strCommand & IIf(Trim$(strOutput_File_Path_And_Name) = "", "", " /O" &
    strOutput_File_Path_And_Name)
strCommand = strCommand & IIf(Trim$(strParameters) = "", "", " /P" & strParameters)
strCommand = strCommand & " " & strAdditional_Options

'NOTE: THE 'HALT' MANIPULA COMMAND WILL MAKE THE MANIPULA
'PROGRAM APPEAR TO FAIL GIVING US A WAY TO GET A MESSAGE
'BACK FROM BLAISE ABOUT WHAT HAPPENED.

'NOTE: lngAppReturnValue HAS A VALUE GREATER THAN 0 IF WSHSHELL
'IS WAITING FOR THE APPLICATION TO END AND THE APPLICATION HAS
'FAILED IN SOME WAY - USE lngAppReturnValue TO GET RETURN VALUE FROM WSHSHELL
Dim lngAppReturnValue As Long

lngAppReturnValue = WshShell.Run(strCommand, 3, True)

If lngAppReturnValue > 0 Then
    'CODE FOR FAILURE OF COMMAND CALL
    MsgBox "The Manipula call: '" & strCommand & "' failed for some reason. " & _
        "Please notify programmer."
    Exit Function
End If

RunManipulaProcess = True
Exit Function
ERR_HANDLER:
    Call ErrorHandler

End Function

```

```

*****
'PURPOSE: TO EASILY RUN DATAMODELS (DEP.EXE) AND CONTROL THE ORDER
' OF EVENTS BETWEEN THE TRACKING SYSTEM AND DEP
'INPUTS: DEP.EXE'S PATH AND FILE NAME,
' DATAMODEL'S DIRECTORY PATH,
' DATAMODEL'S FILE NAME,
' SUBJECT ID TO USE IN DATAMODEL,
' BOOLEAN FOR WHETHER TO EXIT DEP WHEN FINISHED WITH
' SPECIFIED ID'S INTERVIEW,
' MENU FILE NAME IF USING ONE (OPTIONAL)
'RETURNS: BOOLEAN TRUE IF SUCCESSFULLY RUNS DEP PROCESS
'PROGRAMMER: JOHN CASHWELL
'LAST UPDATED: 7.31.2001
*****
Public Function RunDEPProcess( _
    strDEP_EXE_Path_And_File_Name As String, _
    strDatamodel_Path As String, _
    strDatamodel_File_Name As String, _
    strID As String, _
    blnExit_Blaise_When_Done As Boolean, _
    Optional strBlaise_Menu_File_Name As String) As Boolean

On Error GoTo ERR_HANDLER

'WSHSHELL OBJECT
'object.Run(strCommand, [intWindowStyle], [bWaitOnReturn])
'SEE FUNCTION RunManipulaProcess FOR DIRECTIONS
Dim WshShell As IWshShell 'FOR CONTROLLED SHELLING TO DEP
Dim strDriveLetter As String 'TO SET WORKING DRIVE
Dim strCommand As String 'TO BUILD THE FULL DEP COMMAND STRING

'GET DRIVER LETTER
strDriveLetter = Left(strDatamodel_Path, 1)

'INITIALIZE RETURN VALUE TO FALSE
RunDEPProcess = False

'CREATE THE WSHSHELL OBJECT
Set WshShell = CreateObject("Wscript.Shell")

'CHANGE DIRECTORY TO WHERE THE MAN AND BLAISE DATA RESIDES
If Dir(strDatamodel_Path, vbDirectory) = "" Then
    MsgBox "The directory specified for " & strCommand & _
        " does not exist. Process cancelled."
    Exit Function
End If
'CHANGE DRIVE AND DIRECTORY TO FOLDER THE MAN AND BLAISE DATA RESIDES
ChDrive strDriveLetter
'CHANGE THE DIRECTORY
ChDir strDatamodel_Path

'WSHSHELL OBJECT
'object.Run(strCommand, [intWindowStyle], [bWaitOnReturn])
'SEE FUNCTION RunManipulaProcess FOR DIRECTIONS
'NOTE: THE HALT COMMAND IN MANIPULA WILL MAKE THE MANIPULA
'APPLICATION APPEAR TO FAIL GIVING US A WAY TO GET A MESSAGE
'BACK FROM BLAISE ABOUT WHAT HAPPENED.

'BUILD THE COMMAND STRING FROM THE OPTIONS PASSED IN TO FUNCTION
strCommand = strDEP_EXE_Path_And_File_Name
strCommand = strCommand & " " & strDatamodel_File_Name
strCommand = strCommand & IIf(Trim$(strID) = "", "", " /G /K" & Trim$(strID))
strCommand = strCommand & IIf(blnExit_Blaise_When_Done, " /X", "")
strCommand = strCommand & IIf(Trim$(strBlaise_Menu_File_Name) = "", "", " /M" &
    strBlaise_Menu_File_Name)

```

'NOTE: THE 'HALT' MANIPULA COMMAND WILL MAKE THE MANIPULA
'PROGRAM APPEAR TO FAIL GIVING US A WAY TO GET A MESSAGE
'BACK FROM BLAISE ABOUT WHAT HAPPENED.

'NOTE: lngAppReturnValue HAS A VALUE GREATER THAN 0 IF WSHSHELL
'IS WAITING FOR THE APPLICATION TO END AND THE APPLICATION HAS
'FAILED IN SOME WAY - USE lngAppReturnValue TO GET RETURN VALUE FROM WSHSHELL

Dim lngAppReturnValue As Long

lngAppReturnValue = WshShell.Run(strCommand, 3, True)

```
If lngAppReturnValue > 0 Then
    'CODE FOR FAILURE OF COMMAND CALL
    MsgBox "The call to " & strCommand & " failed for some reason. " & _
        "Please notify programmer."
    Exit Function
End If
```

```
RunDEPProcess = True
Exit Function
```

```
ERR_HANDLER:
    Call ErrorHandler
```

```
End Function
```

Here is some sample front-end code showing the above classes in action.

```
'THIS CODE RESIDES UNDER A VISUAL BASIC FORM WHICH DISPLAYS
'A DATAGRID OF SUBJECTS TO BE INTERVIEWED AND A BUTTON TO START
'AN INTERVIEW. THE DATAGRID INCLUDES SUBJECT IDS, NAMES, AND GENDERS

Option Explicit

'CLSINTERVIEW OBJECT CREATED TO RUN BLAISE INTERVIEWS
Private moInterview As New clsInterview
'MSTRID IS A USED TO STORE THE CURRENT ID NUMBER SELECT IN A DATAGRID
Private mstrID As String

'THIS ROUTINE OCCURS WHEN THE SUBJECT CLICKS THE 'START INTERVIEW' BUTTON
Private Sub cmdInterview_Click()
On Error GoTo ERR_HANDLER

    'STRIMPORTSTRING IS FOR CREATING THE DATA STRING THAT WILL BE
    'PASSED TO MANIPULA (SUBJECTID+NAME+GENDER)
    Dim strImportString As String

    'MSTRID IS FILLED WITH COLUMN DATA FROM THE FORM'S DATAGRID (DGSUBJECTS)
    mstrID = Trim(dgSubjects.Columns(0).Text)

    If mstrID = "" Then
        MsgBox "Please select a subject."
        Exit Sub
    End If

    'WHEN THE END USER CLICKS THE START INTERVIEW BUTTON,
    'DISABLE THE FORM BUTTONS FOR STARTING THE INTERVIEW AND EXITING THE FORM
    cmdInterview.Enabled = False
    cmdClose.Enabled = False
    With dgSubjects
        strImportString = mstrID & "StudyID"
        strImportString = strImportString & "," & .Columns(1).Text & "fname"
        strImportString = strImportString & "," & .Columns(2).Text & "lname"
        strImportString = strImportString & "," & .Columns(3).Text & "gender"
    End With

    'CALL THE clsInterview METHOD DO RUN INTERVIEW #1, PASSING IN THE ID
    'INPUT FILE'S PATH (FYI, THE INPUT FILE IS EXPECTED TO BE NAMED WITH THE SUBJECTID
    'i.e. 1000.in), AND THE DATA STRING TO BE IMPORTED INTO MANIPULA

    If moInterview.StartInterview(1, mstrID,
        "K:\PROJECTS\CATI\CPHRE_TRACKING\BLAISE\INPUT", strImportString) Then

        'IF THE STARTINTERVIEW METHOD RETURN 'TRUE' THEN RUN CLSINTERVIEW'S
        'GETRESULT METHOD AND STORE THE RETURN VALUE IN THIS FORM'S DATAGRID.
        'IT SHOULD RETURN 'C' FOR COMPLETES AND 'P' FOR PARTIAL COMPLETES.
        'GETRESULT EXPECTS YOU TO PASS IN THE OUTPUT FILE DIRECTORY AND FILENAME.
        'THAT OUTPUT FILE IS CREATED BY A MANIPULA PROGRAM (END.MAN) WHICH IS CALLED
        'IN THE CLSINTERVIEW.STARTINTERVIEW METHOD (FYI, THE OUTPUT FILE IS EXPECTED TO BE
        NAMED WITH THE SUBJECTID, i.e. 1000.out)
        dgSubjects.Columns(4).Text =
        moInterview.GetResult("K:\PROJECTS\CATI\CPHRE_TRACKING\BLAISE\OUTPUT", mstrID &
        ".OUT")
        dgSubjects.Refresh
    End If

    'WHEN THE MANIPULA PROGRAMS AND BLAISE INTERVIEW ARE FINISHED PROCESSING,
    'ENABLE THE FORM BUTTONS FOR STARTING THE INTERVIEW AND EXITING THE FORM
    cmdInterview.Enabled = True
    cmdClose.Enabled = True
    Exit Sub
ERR_HANDLER:
```

```
Call ErrorHandler
End Sub
```

It is worth noting that a programmer can use the HALT command in Manipula programs to prematurely end them when something is not correct. If the Manipula program is called with Microsoft's Windows Host Script, an error level code is automatically returned when a HALT is processed. This allows the Visual Basic programmer to use the error level code to notify the end user that an error was encountered and the interview can be cancelled. Here is a snippet from a Manipula program where the HALT command is used.

```
...
MANIPULATE
  infile.READNEXT

  if (infile.StudyID<>PARAMETER) OR
    (infile.Gender=EMPTY) OR
    (infile.FNAME=EMPTY) OR
    (infile.LNAME=EMPTY) THEN

    HALT

  else
...

```

It is important to understand that Microsoft's Windows Host Script allows the Visual Basic application to be used while it is processing other requests. The only place the code waits for the external process to end is inside the function where the WshShell object is called. This means that the programmer can be assured that the code within the function containing the Windows Host Script object will operate synchronously. However, the programmer must alter the properties of the form as needed to control the user's choices while running a process such as a Blaise datamodel. In the example above, the Start Interview and Exit Form buttons are disabled so the end user cannot exit the form or start another interview while an interview is still being processed. When the interview is finished the button are enabled.

CONCLUSION

In conclusion, there are several ways to synchronously shell between Visual Basic and Blaise. However, Microsoft's Windows Host Script seems to be the best solution. It require only a few lines of code, is easy to understand, works across Windows operating systems when calling Manipula or Dep, frees the end user to continue working in the tracking system, and receives error level codes from Manipula and Dep.

Standardised Survey Management at Statistics Netherlands An Example of the Object Approach

Frans Kerssemakers, Statistics Netherlands

Marien Lina, Statistics Netherlands

Lon Hofman, Statistics Netherlands

1. Introduction

Data collection for household surveys at Statistics Netherlands has since long been concentrated in one specialised department. With the arrival of laptop-computers for multi-survey use around 1990 this department got an interview administration system (IAS). IAS was centred round a central database for all CAPI surveys, storing samples and interviewers, with automated tools for supporting the allocation of cases to interviewers and providing them with material, keeping book of the cases sent back (by phone), checking on (piecework) payments and reporting on progress and results. For decentralised case management on the interviewer's laptop a dedicated system had been developed in the early nineties called LIPS (see Hofman and Keller, 1993), which took care of receiving and handling cases and preparing them for transmission by phone. It contained questions for the interviewer about how a case was dealt with, mainly concerning home visits and their results. Later on a standard Blaise questionnaire was used for the interviewer's reporting, next to one or more data models for the actual interviewing, all these embedded in a self-processing Manipula program, using Maniplus. Separate from CAPI the centralised CATI facility had its own Blaise-integrated management system for call scheduling, making appointments etc.

Top-down automation of case management for CAPI appeared to be a huge effort at the time. But once the system went ahead, it could serve most surveys till the end of the nineties. Then fighting non-response in particular necessitated more versatile designs such as mixed-mode and longitudinal ones. Although multi-wave surveys had originally been taken into account potentialities were insufficiently used and tested to get at a self-supporting system. In a situation where so many things had to be rearranged at once the system first had to establish itself for the then prevailing one-time surveys. Besides, being relatively inexperienced, it was not the right moment to develop a quality plan for orderly maintenance and expansion of the system to meet future growth. Later on, as a consequence of several reorganisations, adaptability began to suffer from staff turnover and loss of experience-based knowledge. As new researchers often were afraid to jeopardise ongoing data-collection, this concomitantly resulted in a gradual loss of control over the more managerial aspects of surveys. Automation specialists on the other hand often had to take decisions unwillingly, without knowing much of the research background, just to have the system operating.

The functioning of case management became rather critical around 2000 when a longitudinal design was mounted for the Labour Force Survey (LFS), using CAPI the first time and CATI for four subsequent waves. Because the old management system could not meet the needs of the new design, as was the case with several other surveys, a hybrid situation arose in which increasingly survey-specific systems were being developed while at the same time maintenance of the still indispensable old system was becoming more and more cumbersome. From this and under considerable time pressure an urgent demand for redesign arose. In accordance with the bureau's tradition a *standardised approach* for all surveys was again judged the best solution, but starting point this time should be the ability to deal with different waves and different modes in the same survey, so as to permit dedicated designs for specific demands. The question then became which elements were general enough to be standardised and to what extent flexibility should be built in.

Simultaneously plans were evolving for a new fieldwork organisation in which CAPI interviewers were to be regular employees instead of being hired as freelancer. It was intended to have more supervision and coaching and closer working together in local groups. One of the goals was to get

better opportunities for *flexible solutions* particularly for interviewer deployment problems and non-response. This would increase the need for exchanging messages electronically and for moving around cases between interviewers. At the same time we wanted the central office to be well-informed on the progress of the fieldwork, enabling the researcher or field staff to adjust in time such things as the workloads assigned to the interviewers, the sampling plan or other parts of the design. It was therefore decided to channel all relevant information to a central database from which the latest info should be readily available to all interested parties. This means, for instance, that a local supervisor can be the one who decides to re-allocate an individual case. The transmission of the case from one interviewer to another, however, should still run through the central system.

2. A pragmatic approach to the new design

Thus we were faced with the necessity to redesign case management in the short term, making it fit for handling more complex mixed-mode and multi-wave designs, in a way to make it easily adaptable (regarding anticipated but partly uncertain future developments), all this within a standard framework.

Putting the information for case management on all surveys in one central database and having this information available for multipurpose use makes the development of standard definitions and routines indispensable. Sharing the same database implies the moulding of information on cases in a common set of variables or fields. Yet not every field has to be filled for every survey. So there is room for adding specific pieces of information for particular surveys. Also, at the front-end things may vary between surveys, for instance the reasons for non-response displayed to the interviewer, while at the back-end a common non-response variable may be derived for comparison.

We did not start from scratch. Having gained experience with case management for many years we had already developed ideas for improvement. We were also used to standard routines and question blocks for case handling by the interviewer and for standard tasks like listing or rostering of household members. So the new design could partly be based on the problems we had met and partly on tightening up existing practises. But we also had to enlarge the possibilities of the new system to make it ready for the foreseeable future. From this the following starting points were chosen:

- Surveys are to be considered multi-wave surveys and provisions should be made to pass on data from one round or period to the next. Also within a one-time survey there can be successive approaches by using different interviewers or modes (especially regarding non-response).
- It should be possible to use different modes concurrently (as respondents may express a preference for self-completion on paper or through Internet instead of being interviewed).
- Having a standard framework for all surveys while at the same time valuing certain statistically important distinctions can best be handled by explicitly defining these distinctions and have them being parameterised (specifying the characteristics in the context of the case management system).
- As far as the researcher is responsible for certain design decisions this party should also be in charge of the settings and options which effectuate these decisions.
- Not so much as a matter of principle but for practical reasons we give preference to only one data model for a survey. Above all, this should improve the researcher's access to the specs of the data model and through increased transparency simplify maintenance of ever changing questionnaires.

Access and maintenance have proven to be major issues. At the IBUC-conference in 1997 Statistics Netherlands presented an integrative perspective for a Continuous set of Surveys on the Quality of Life, called POLS (Heuvelmans et al, 1997). The characteristic feature of this design is the use of a common basic questionnaire and different data models for specific subjects (next to a separate data model for the interviewer's reporting on the case). Although representing quite a logical structure it soon came out that the dependencies between data models and the concomitant exchange of data put high demands on technical expertise and staffing. As we were already warned of at the same conference this increased 'the reliance on programmers for complex elements of questionnaires, and on non-Blaise software' (Manners and Deacon, 1997) and made the design vulnerable to staff turnover.

For researchers, who are not always Blaise experts after all, it generally became more difficult to get access to parts from other data models which might be relevant to them and, especially in a process of change, to lift out their own question modules for developing purposes.

3. Key elements of the design

3.1. *The researcher back in charge*

As a reaction to the difficulties encountered we have now chosen to avoid the concurrent use of several data models for the same interview case (leaving aside external data models that only contain a description of data to be read by the questionnaire program). This choice has been made possible by the increased possibilities of both Blaise and hardware. Using one program, the best place to account for the way a household has been approached and to report on the achieved results are usually *parallel blocks* (cf. the use of Non-response- and Appointment-blocks in CATI). The special function of such blocks can be displayed to the interviewer by putting them on tab sheets next to the actual interview part. In this way meta on handling the case can easily be built in the same data model used for interviewing, as we did in a block called *Admin*. Tracking down persons or households that appear to have moved takes place in a similar way in a block called *Moving*. Apart from solving technical and organisational problems, we believed it important to strengthen the researcher's control of parts that provide meta information on a case, in order to allow monitoring how the sample is finished off or studying how distinct approaches and various interviewers produce different results. Because these parts were previously applied in a standard way by specialists outside the project team, the researcher usually did not feel in control. Belonging to the same data model also makes the exchange of data with the interview part easier. This does not mean that every researcher will from now on fully understand what is in the source code. What makes a difference, however, is the notion that all specifications which may effect the outcomes are in the same data model that the researcher has access to, and are written in ordinary Blaise.

3.2. *Data exchange with Blaise*

Almost by definition every survey has its own distinct questionnaire. Concentrating data-entry entirely in one data model can therefore easily lead to very survey-specific solutions. Standardisation of case management on the other hand asks for standard blocks in the questionnaire to supply the central database with the demanded data. As far as data exchange is concerned, however, it is better to speak of *standard fields*. For the only thing the central database needs to know are the names and the types of the fields from which to read the input variables. The link between the Blaise database and, in our case, the SQL server database is made using the Blaise API as 'communication tool'. In case the standardised blocks are nested in other blocks also the names referring to these blocks are needed. But apart from the standard fields themselves, blocks like *Admin* or *Moving* may well differ between surveys and can be adapted or extended in many respects whenever needed.

3.3. *Standard blocks*

Whether different surveys also use the same *standard blocks*, i.e. blocks of the same type, is a matter of policy. We prefer to have common standard blocks for well-defined subjects, but at the same time may allow different surveys to use different parts of it. This forces us to explicitly define on which grounds a survey deserves a different treatment. For instance, we consider the listing of a household from scratch a more demanding task for the interviewer than checking an already existing list for changes. We will consequently use two different blocks within our standard module for household rostering. Standard then refers to common solutions for the same type of problems, such as using the same fields if possible (kinship, marital status etc.) or composing the same type of output for the central database e.g. a standard block with the gender, age, etc. of a person and whether this person is a target, contact or proxy (i.e. the row in the household table or matrix).

3.4. The survey definition

For a differential treatment the standard blocks should know what type of survey they are dealing with. Who can better specify this than the researcher in charge of the survey? It is therefore the researcher who should provide the case management system with specifications of the survey or, as we call it, the *survey definition*. Many characteristics of a survey have to be known to the central system beforehand. For the allocation of cases a fieldwork period has to be set and if there is data from a previous period to be sent to the laptops the system should know. But also whether it is allowed to skip one period of measurement, for instance. So it is rather obvious to define a survey beforehand at the level of the central system, including its features with respect to the way standard blocks are used.

A clear distinction then arises between the researcher who defines the survey and is in charge of the data model for the questionnaire and the IT-staff responsible for the proper functioning of the case management system given the definition of the survey and the relevant standard blocks in the questionnaire. Of course, there should also be an initial sample of cases, whereas an optimal deployment of interviewers has to be based on up-to-date and detailed information on their service contract, training, optimal and maximum workload and availability.

3.5. An object-based approach

Once a case is transmitted to a laptop we deliberately choose to conduct data-entry for this case as much as possible within the questionnaire, irrespective whether it concerns data from interviewing or meta data on the case provided by the interviewer. Where cases from different types of surveys should get a different treatment using the standard blocks, this is pre-specified in the survey definition. This implies that outside the questionnaire data-entry on the laptop should be reduced to a minimum. As already mentioned in the introduction, Statistics Netherlands is also increasingly facing complex and changing designs. We therefore badly need a more flexible approach towards case handling, moving individual cases from one interviewer or mode to another (instead of assigning more static workloads as we were used to). Questionnaires too have to be better geared to changing demands, for instance when they are used to provide information on gaps in other sources, like registers (the use of which is in our case rapidly increasing). This is where the object-based approach comes in.

As the principles of an object-based approach to case management are extensively dealt with elsewhere (Gray, 1995) it suffices to say that every case can be represented by an independent object. The power we want to use rests on the 'ability to hide internal details from systems that don't need to know them' (Gray and Anderson, 1996). Specific details, like the data collected, are fully encapsulated. This makes it possible to abstract from survey-specific content. An object needs a method that tells the system what to do when the object is activated. From this perspective sample cases can be considered as objects to start questionnaires (and the latter in their turn as objects to install questionnaires). There should also be a 'caption' that makes visible what the interviewer needs to handle the object before it is being activated (e.g. address info and the name of the survey).

3.6. The new LIPS

The assistance our newly developed case management system gives to interviewers for handling objects on the laptop is largely based on the Casebook system from UK's Office for National Statistics. ONS also offered great help with adapting the system to our fieldwork organisation. Their object-based approach shaped our view on how the case management processes should be re-arranged.

Between receiving a case and starting the questionnaire the interviewer should be assisted in adding the object to the workload and keeping an eye on the progress made in dealing with the workload, facilitated by various ways for conveniently arranging the cases from different surveys. When the interviewer wants to take a look at a particular case things like the survey and period and address of a

household or name of a person should be displayed. From this screen it should also be possible to start up the questionnaire.

A mechanism was built to get direct access to the parallel blocks *Admin* and *Moving* (by pushing the corresponding buttons). For it may be confusing to enter the interview part of the questionnaire form if one just wishes to book some visits (in *Admin*). We also created a facility to display the content of a special field in *Admin*, which may be used for a remark (comparable to the display of a selected field on the dial screen in CATI-management). Finally the screen is used to finish cases. When choosing this option an action is started by which a case will be marked 'dirty' if not completed the proper way. A message will then be displayed indicating what piece of information is still missing, e.g. the final result has not yet been reported (by the interviewer). The checks on completeness, however, are executed in the questionnaire. There the error message is computed and stored as a string. The action to finish a case only reads from the questionnaire whether '*ready = yes*' and, if not, the error message. A reported error can be overruled and the 'dirty' case be moved to the out-tray by the interviewer, putting it ready for transmission to the central office. There its erratic status will be noticed and special attention will be given to the interviewer and the case, as being a possible candidate for re-allocation. Together with the case a progress report is sent to the central office concerning the workload of the interviewer still to be finished. In a similar way the report is based on reading a field in *Admin* in which the interviewer has to report (in a very simple way) on the intermediate results achieved.

A lot of actions are taken in the new LIPS, partially induced by the interviewer making choices what to do next. However, in the new LIPS the interviewer is not answering questions, not even whether a given address can be found, or whether anyone is living there. All this is reported in the questionnaire. The only exception is for extra households that may be found at a given address. This only applies to a first-time approach when we use a sample of addresses to get at a sample of households. Under certain conditions the interviewer can put on stage an extra household, up to three households in total. If permitted to do so a special button will appear on the start-up screen. For case management, however, the assigned case still remains the complete address. To be sent back as 'clean' all households within it must be finished. But if cases should be re-approached in a subsequent period then the households are the sample elements to be followed (as we generally do not use area sampling). The central system will then replace the initial address by one case for each newly identified household.

3.7. Panel management on a flow basis

Once a survey has been defined the system should know what to do in each instance for every case. In longitudinal or multi-wave surveys we strongly believe that the central system should automatically select cases for the next wave or period on the basis of the Blaise data received. That is to say, the Blaise database should preferably contain all necessary information for the system to decide on this issue. Generally, when a refusal has been derived in the questionnaire, reading this result from a standard field will automatically cause the non-continuation of a case, although it may lead to some other action like selection for refusal conversion. In addition to this standard mechanism a researcher may have specific reasons for removing a case from a panel group, for instance, if he wants a complete data set on all targeted persons in a household. Very often these kinds of conditions can be pre-specified by the researcher in the data model and the possible blocking code put ready for reading by the system. We also want data inconsistencies to be solved as much as possible by the interviewer, making conflicting information visible through the questionnaire. From our experience there is little need for data-entry outside the questionnaire form, even with complex panel studies (except for change of address cards). So with panel surveys our central system serves to pass on cases to the next period, mostly together with input data from the previous period that has to be checked for changes.

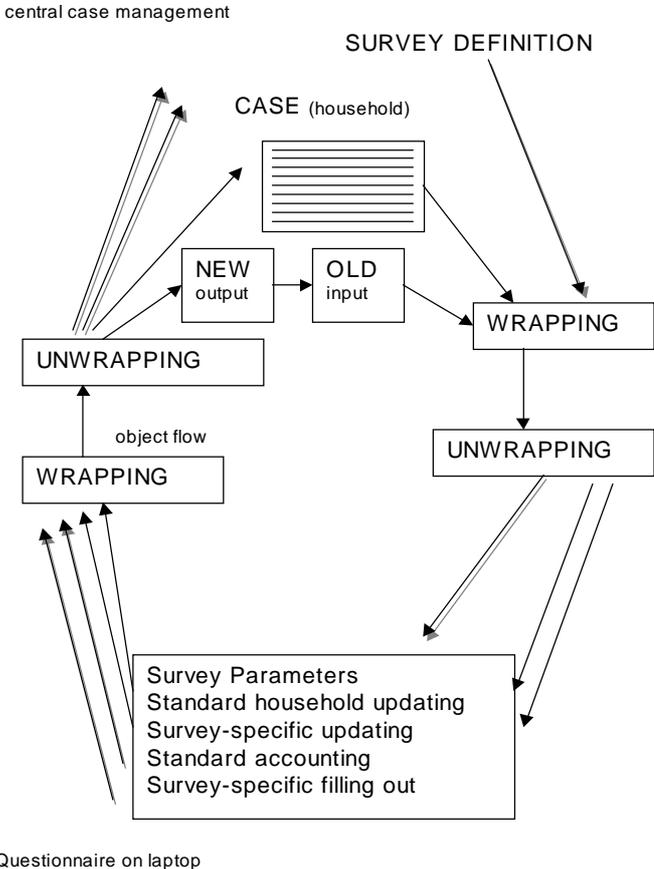
Because our cases are usually households or persons within households we choose to represent each case by way of a matrix in which the rows are used for the household members. This conveniently fits in with the standard matrix we use in the interview form for listing the household members. In terms of variables or fields the row is identical to the block representing a person in the interview form. A standard block for each person can therefore easily be copied both ways. For this we use the Blaise

API. When it is found during fieldwork that a household has split up and all separate parts have to be approached again then the rows only have to be divided to form new cases. Within each row the status of the corresponding person with regard to the survey can be specified. For instance, a certain child may be the target person, the parents the proxy-persons and one of them may be the one to whom the advance letter should be sent. Linked to the matrix is a survey-independent id-number and info on the name, address or telephone number. In the near future the matrix will probably be filled beforehand by combining information on persons and where they live from the Dutch Population Register. Then the interviewer only has to check this information and adjust it if necessary.

Finally, there will often be survey-specific information to pass on. Again, we want the data to be preloaded in the questionnaire form itself. The required Blaise data set should be in a block. This block, although specific to each survey, should be referred to by a standard name, NEW, to indicate the updated output of the panel wave. The central system will look for the name referring to the block. It does not process the data in any way. It just copies the variables in it to an identical block in the data model for the next period (using the Blaise API, and possibly a conversion tool in case the data model has changed). This block, named OLD, is of the same type as NEW. Whereas NEW is meant to deliver the data for the next period, OLD contains the input data for the new panel wave. So the output is automatically made input, for each case to be re-approached.

Generally conceived, as sketched out in figure 1, we discern two circular motions in case management: One for the updating of standard fields and the other for the updating of survey-specific fields. Besides, every fieldwork period has its own survey definition (although usually not changing very much). This definition is made available to the questionnaire by the new LIPS in the form of external data (which after reading serve as survey parameters). And, of course, there are the one-time outcomes, part of which is used for monitoring and evaluation of the case handling.

Figure 1. Flow of panel data



4. The role of the SQL server database

To reconcile differences between a more rigid standard approach and more flexibility we first had to agree on a minimal set of functional requirements. To the functionality which was already fully realised in the old system the following major extensions were added:

- Comparability on (non-)response results across surveys on the base of a common variable with standardised categories (the score on which is derived from questions displayed to the interviewer on the laptop, which may differ from survey to survey)
- Automatic computation of various standard response measures and the production of standard reports for different purposes (statistical, accounting, checking on performance etc.)
- Storage and accumulation of collected data on reasons for non-participation (refusal, no contact, no opportunity etc.) to be used for non-response analysis
- Quick reallocation within the same fieldwork period of individual cases from one interviewer to another, including the transmission of Blaise-records already partly loaded with data from previous periods or records containing external data
- Easy to handle facilities for shifting a case from one mode to another, e.g. from CAPI to CATI to increase the probability of contacting, and if contacted, e.g. from CATI to CAPI to complete the larger part of the questionnaire; or from interviewing to self-administering a questionnaire form, offering varying degrees of support
- Registering the use of sub-forms in a CAPI- or CATI-survey, such as a self-administered question form, to be collected by the interviewer or sent back directly to the central office

For the central system to know how to handle a case we had to define a number of general entities common to every survey. These include:

- The identification of the survey and the particular fieldwork period for which cases are selected
- The version of the questionnaire to be used when data collection on a case is started
- The definition of the survey, given a standard approach, to allow for a differential treatment; for instance, when handling a first-time sample of addresses versus a persons' sample
- The different tasks within a survey to be discerned, like allocation, re-allocation or changing the mode of data collection, special coding, processing the incoming data and possibly changed address info, putting the eligible cases ready for the next period, wrapping cases in objects, at the same time keeping up-to-date the stage of the process where every case is in
- A standard set of categories to represent the final result and the home visits or phone calls made
- The identification of the case and mostly also of the persons making up a case (see section 3.7)

5. The new LIPS: some technicalities

As mentioned in section 3.6 the design of the new LIPS is based on the ONS object approach. We decided to start the development of the new LIPS from scratch just using the ideas from ONS. We invited one expert to visit us for one week. The idea was to try to build a prototype to prove an object-based LIPS satisfying the needs of Statistics Netherlands is possible. The prototype could form the basis for a final implementation. After some discussion and based on the experience with the old LIPS system and the experience from ONS we made some important implementation decisions:

- We decided to use Maniplus as the main tools to build the prototype. The only alternative was Microsoft Visual Basic in combination with the Blaise API. But to keep things simple, both development and laptop installation, we decided to prototype in Maniplus.
- We decided to use only 'modern', Windows based technology when we have to go outside Maniplus because of missing functionality. So no use of batch-files, 16-bit DOS tools (like PKZIP) and shelling-out to the operating system (for instance executing a MKDIR command to create a new folder). Experience with the existing system indicated that using such old-fashioned 'DOS-based' solutions can introduce instabilities and will lead to less over-all control. A good choice as Maniplus-extender proved to be VB-script. Within VB-script it is possible to use numerous available ActiveX objects to handle specialised functionality. For instance for creating

ZIP-files we selected an ActiveX archiving tool. With VB-script in combination with the ActiveX archiving tool it is possible to have total control over the archiving process.

- We decided to use password protected ZIP-files as physical implementation of the objects. These ZIP-files always contain at least a file called object.ini. This file has a simple so-called ini-structure. Ini-files are very flexible in use and can easily be extended with new sections and new keys without causing compatibility problems. All information on how the object has to behave is present in the ini-file, but the ini-file can also be used by the object to store extra information, like progress information on an assignment of a specific address.
- We decided to make the system completely data model independent. So all object methods implemented are independent of data models used for interviewing. Compared to the old system this means for instance that no specialised, data model dependent Manipula set-ups are used to extract data from the interview or to pre-load data. To make it possible to use data from an interview in LIPS, a way had to be found to extract data from a Blaise data file without knowing the data model while preparing the Manipula LIPS set-ups. The Manipula function GETVALUE was added to the Blaise system to achieve this goal.
Because the system does not know the data model, it is also not possible to create a Blaise form run-time in the field by the Manipula LIPS application. A form can in the new LIPS only be created by the data entry program (by passing the correct command line parameters to the EDIT function in the Manipula setup) or before hand at headquarters.
So in the new LIPS all data required by the interview need to be included in the object. There are roughly two possibilities:
 - The data is stored in the object.ini file as ASCII data and made available by LIPS to the interview session as external ASCII file. For instance all survey definition related data is passed on this way.
 - The data is preloaded in a Blaise form at headquarters and included as file inside the object. For instance when panel related data is present a form is created at headquarters.
- In case of an address sample multiple forms (each corresponding with one household) can be required at one address. In this case the Blaise form is created by passing a unique identifier as key via the command line to the interview session. In case the sample contains only identified elements (for instance a person sample based on the Population Register or a panel follow-up) a form is created at Statistics Netherlands with the correct data using the Blaise API.
- As described in previous sections all administrative data collected in the field is part of a standardised parallel block of the data model. To allow the interviewer easy access to this parallel block from within LIPS a new command line option has been added to the data entry program. By mentioning the correct parallel name on the command line the data entry session starts with the focus on that parallel. The name of the parallel to be started is stored in the assignment object.

5.1. Working with the new LIPS

The new LIPS-application uses three so-called trays: The in-tray, the work-tray and the out-tray.

The in-tray

In general objects mailed to the laptop arrive in the in-tray. (LIPS can also handle so-called ‘Auto activate’ objects, for instance for installing new software on the laptop. We will not discuss this any further). The interviewer sees in the in-tray a short description of the contents of each object, for instance ‘New assignments for the Labour Force Survey, period June, 2001’. The only action available for the interviewer in the in-tray is ‘Accept’. This action moves the object from the in-tray to the work-tray.

The work-tray

The work-tray contains all objects the interviewer can activate. A number of different object-types are available (and new object-types can easily be added to the system). The most important object-types

(from the interviewer perspective) are: the ‘Install questionnaire’ object, the ‘Assignments’ object and the ‘Individual assignment’ object. What happens when the object is activated depends on the object-type. For instance, activating the object with the new assignments for the LFS will result in extracting the individual LFS assignments to the work-tray and a self-destruct of the LFS assignments object. Extracting the individual assignments is only possible when the correct LFS questionnaire has been installed on the laptop. In case this questionnaire has not been installed an appropriate error message is displayed.

If the object allows it, the interviewer can move the object from the work-tray to the out-tray. For instance, an ‘Install questionnaire’ object can not be moved, but ‘Individual assignment’ objects can be moved to the out-tray.

The work-tray shows for each object a short description plus the contents of a special entry in the *object.ini* file. This entry can be used for instance to show the progress made with the assignment (for instance ‘No contact’). The contents of the work tray can be sorted based on the contents of some entries in the *object.ini* file. The display order of the objects on the screen is always based on the object type. Because of this objects of the same type are grouped on the screen.

The out-tray

The out-tray contains all objects that need to be mailed to headquarters. The interviewer can move an object back from the out-tray to the work-tray. During data communication all objects present in the out-tray are mailed and, when this is successful, removed from the out-tray. During data communication administrative information is extracted from each object present on the laptop (so from all trays!) and mailed to headquarters. This information can be used to monitor the progress in the fieldwork. The content of this information depends on the object type.

Activating the individual assignment object

When this object type is activated the interview method is activated. In general this is another Maniplus setup that is responsible for handling the assignment. The main functionality offered is:

- Detail information on the assignment (like the full survey name, survey period, full address information, etc)
- The possibility to spawn extra households (if applicable)
- The possibility to start the interview
- The possibility to start the accounting parallels directly
- The possibility to check whether the assignment is completed. This is done by inspecting the contents of one field in the interview form that summarises the result based on collected interview data and data filled out in the accounting parallel block.

6. Some API technology

On the other side of the interview system, records filled with interview data are coming in from CATI or CAPI and must be linked to the newly developed central interview administration system. The management software that handles this is built with the Blaise API. The organisational scheme is that packages are entering the system, containing interview data and a status file for each case that had been put out earlier by LIPS. The system determines the state the received data are in, discriminating between *response*, *partial response* and “*not-opened-at-all*”. After that the system takes several decisions, based on the results in the interview record and the status that is in the guiding object file. It would be too much to explain all the details. For example, the system decides to (whether or not) put the record through to the next wave, to a specific interview mode, to update address information in the administration, to create a new record if only one person of a household moves to another place, and many things more. Only cases with unexpected results, such as data corruption, are copied as received to a reserved place where manual intervention is required. By applying certain standard blocks the system can read values in standard fields to recognise the subsequent steps for each case. The entire system has been developed in Microsoft Visual Basic using the Blaise API. The processing time of the

Blaise API approach may be slightly more compared to the old Manipula approach, but the advantage is that there is no need to re-prepare Manipula set-ups for different data models. When data models change (which happens on a regular basis) then the system does not need to be maintained as long as the called fields in the standard blocks remain the same.

The API makes it possible to read information from different data models with the same API application. In this way the API puts aside the problem of data models being incompatible, as long as the data models use a uniform block structure and fields that are used by the Visual Basic application.

7. Conclusions

It is too early to draw final conclusions because the system is not yet in full production. It will go live in January 2002. But, during the implementation and testing of the new system we already noticed that the approach that has been chosen is flexible and it was very easy to make required changes. We are confident that we have chosen the right approach for the years to come.

References

Gray, J and Anderson, A. The data pipeline: Processing survey data on a flow basis, in: *Survey and Statistical Computing 1996. Proceedings of the Second ASC conference*, London, United Kingdom, 1996

Gray, J. An object-based approach to the handling of survey data, in: *Essays on Blaise 1995* Statistics Finland, Helsinki, Finland, 1995

Heuvelmans, F, Kerssemakers, F and Winkels, J. Integrating Surveys with Blaise III, in *Proceedings of the Fourth International Blaise Users Conference* INSEE, Paris, 1997

Hofman, L and Keller, W. Design and management of Computer Assisted Interviews in the Netherlands, in: *Journal of Official Statistics*, 9, 1993

Manners, T and Deacon, K. An integrated household Survey in the UK, in: *Proceedings of the Fourth International Blaise Users Conference* INSEE, Paris, 1997

Survey of Labour and Income Dynamics (SLID) Project at Statistics Canada

Richard Chan
Operations Research and Development Division
Statistics Canada

Background

Statistics Canada has been conducting CATI social surveys for a long time. However, most of the surveys are not truly CATI. The interviewers just phone the respondents and conduct a CAPI survey. There is no callscheduler to handle the distribution of cases. Other CATI surveys are not in Blaise. Recently, the bureau decided that all CATI social surveys should be implemented in Blaise with callscheduler to provide full CATI functions. As a result, we have had to redesign the infrastructure and the survey applications to accommodate all the changes.

Survey of Labour and Income Dynamics (SLID) was chosen to be the first survey implemented in full Blaise. It is a longitudinal survey designed to follow the same respondents for several years to get a long-term picture of how changes affect people financially. Interviews for SLID are conducted twice a year, in January and in May, for everyone in the sample who is aged 16 or over. Like many other social surveys, SLID consists of Entry and Exit questions, Demographic and Relationship questions, and subject matter questions. In January, the subject matter is Labour. Currently, some of the social surveys are implemented in Visual Basic and Blaise. The Demographic and Relationship part is done in Visual Basic, and the subject matter component is done in Blaise.

Design

The redesign of the SLID application is a challenge. Not only do we have to replicate the functionality of the existing application, we are also required to replicate the “look and feel”. Since the survey is divided into three parts, there are two possible designs for the application. One is the single datamodel approach, where we combine all three parts into a single datamodel, like a typical Blaise application. The other is the multi-datamodel approach, where we keep the three parts separated, and use three separate datamodels to implement a single application.

The single datamodel design is obviously the easiest way to implement the application. Technically, it will capture all the required information for the survey, so there is no problem in terms of functionality. However it does not quite satisfy the “look and feel” requirement. Currently, the Visual Basic portion of the application displays a list of members who are the candidates for the Labour component at the end of the Demographic and Relationship (DemRel) component. This list is called the component list. In general, it contains all the subject matter components for each household member. In the case of SLID, there is only the Labour component for each member. The interviewer can select a member from the list and start the Labour component. When the Labour component of that member is finished, the application will return to the list and the selection process is repeated until the interviewer is ready to stop. After that, the application will proceed to the Exit component. The other concern we have is the size of the datamodel. Since we have a roster of forty members and each of them could be a candidate for the Labour component, the datamodel would contain forty Labour question blocks. Even if handling huge datamodels is not a problem, it is still a waste of space. For a household with only one member, we will have one block of information and thirty-nine blocks of empty space.

The multi-datamodel design is more complex, however it provides more flexibility. With this design, we separate the Labour component from the DemRel component. We can execute certain functions before

switching from one component to the other, which makes displaying a component list possible. Since Labour is separated from DemRel, we no longer have to reserve space for potential Labour records. We will have a record in the Labour datamodel when it is necessary. A household with only one candidate for Labour will only have one record in the Labour database. Although we have more than one datamodel, the size of each datamodel is now smaller. With the multi-datamodel design, we can make the datamodels independent of each other. We can modify and prepare one datamodel without affecting the others. It is actually one of the requests made by the social survey clients. Once the subject matter component has been tested and accepted by the clients, that component should not be modified and prepared again. This is not possible by using a single datamodel. If DemRel and Labour are in the same datamodel, we have to re-prepare the whole datamodel regardless of what changes we are going to make. When we separate them into two datamodels, we can modify and prepare the DemRel part without touching the Labour component. The problem with the multi-datamodel design is passing information from one datamodel to another. For SLID, only respondents aged 16 or over have a record in the Labour database. However, the age is stored in the DemRel database. Also, the flow of the questions in the Labour component is based on the responses to questions in the DemRel component. In order to pass information, we will need a shell program to manage the execution of each datamodel, and the extraction of data from one to the other.

Both designs have their advantages and disadvantages. The single datamodel design is easier to implement. We have all the needed information in one database. Extraction and population of the database will be simple and it does not require a complicated shell program to facilitate the communication among databases. It is self-contained, so it may be easier to maintain and support. However, everything has to be done within the Data Entry Program, which does not provide much flexibility, especially in terms of user interface and handling multiple subject matter components. With this design, certain functions such as the component list cannot be implemented, many of the “look and feel” requirements cannot be satisfied. With the roster size of forty, the database will be very large. The multi-datamodel design, on the other hand, is more complicated and is more difficult to implement. It requires a shell program to drive the whole application. It has to control when to call which datamodel, and extract and route data from one datamodel to the others. The shell program has to be carefully coded in order to keep all datamodels in sync. Nevertheless, this design allows us to display the component list between datamodels, to divide the survey application into distinct components, and to treat each component individually. Since both designs will get the job done, and the “look and feel” requirement is a very important aspect of the application, we decided to choose the multi-datamodel design.

Implementation

Like many other social surveys, SLID can be divided into four different parts: Entry, Demographic and Relationship, Subject Matter (in this case, Labour), and Exit. Although modulising the survey gives us more flexibility, we do not want to over do it as it may complicate the implementation. We know that Labour will be a separate datamodel by itself. Entry is relatively small and is followed by Demographic and Relationship. Since some surveys do not capture Demographic and Relationship at all, it is easier to combine the two parts together and call it EntryDemRel. Exit is the last part of the survey, and it will be a separate datamodel. We cannot combine Labour and Exit, because Labour is at the person or component level, and Exit is at the household or case level. Since each household in SLID can have many components, there are two levels for each case, household level, and component level. There should be only one Exit for each case, regardless of how many components it has. There is an outcome code associated with each level, which indicates the status of the case and its components.

Now that we have divided the survey into three datamodels, we have to decide which datamodel should inherit CATI. The datamodel that inherits CATI should be the entry point and the exit point of a case. We need the callscheduler to deliver cases, and assign treatments to cases upon exit. It seems logical to have EntryDemRel inherit CATI, since it is the first part of the survey. However, we do not exit through

EntryDemRel, but rather exit through the Exit datamodel. We have to assign a treatment to a case, or make an appointment, and assign an outcome code whenever we exit a case. Therefore, the Exit datamodel has to inherit the CATI function.

The core part of the survey is now divided into three datamodels called CallSchedulerExit, EntryDemRel, and Labour. The CallSchedulerExit datamodel inherits the CATI functions and is the entry and exit point of the application. It also contains all the Exit questions for the survey. The EntryDemRel datamodel contains the Entry, Demographic, and Relationship questions for the survey. Although CallSchedulerExit is the starting point of the application, it is in the EntryDemRel datamodel where the actual interview begins. This is why the Entry questions are implemented in this datamodel. The Labour datamodel contains all the subject matter questions of SLID. We also need a datamodel to implement the component list. For each household, the component list database stores a list of components and their outcome codes. We call this datamodel ComponentList.

Here is the overview of the application. We start with the CallSchedulerExit datamodel, which inherits CATI. As it only contains the Exit questions, once we have the case id, we use it to call the EntryDemRel datamodel to start the interview while CallSchedulerExit remains open. When EntryDemRel is finished, the ComponentList database will be updated based on the household information in EntryDemRel. The person id of each household member who is a candidate for the Labour component is stored in ComponentList. A list of the members is displayed, hence the required component list. From the component list, we pick one of the members and use the person id to call Labour. When the Labour component of that member is finished, we return to the component list and repeat the same steps for each remaining members. When we exit the component list, we return to CallSchedulerExit. Whether the case can be finalized depends on the combination of the outcome codes for each component of that case. If the case can be finalized, an outcome code will be assigned automatically, and the Exit questions will be asked. Otherwise, the interviewer will have to assign an outcome manually, which can be final or in-progress. The Exit questions will be asked to complete the case, or the interviewer will assign a treatment or make an appointment to exit the case.

Basically, there are two parts of the application. One is within CallSchedulerExit; the other is outside CallSchedulerExit. When we are outside CallSchedulerExit, we are working with EntryDemRel, ComponentList, and Labour. These datamodels have to pass information to the others. The component list is created based on the age of each member from EntryDemRel. The Labour component is activated by the person id in ComponentList. It also needs other information from EntryDemRel, such as the roster of the household. Since Labour is a proxy questionnaire, any member can answer the questions for any member in the household. Certain flags are also stored in EntryDemRel, which control the flow of questions in Labour. At this moment, the only way to read from, and write to a Blaise database and to create a user interface is by using Manipula/Maniplus. Therefore, we need a Maniplus program to call and manage these three datamodels.

The Program Manager

The Program Manager is the Maniplus program that manages the three datamodels. Its functions include:

- call the EntryDemRel datamodel with a specific case id;
- create a new record in CallSchedulerExit, and EntryDemRel if the household splits;
- read from EntryDemRel to get information for creating the component list of the case;
- update the ComponentList database using the information from EntryDemRel;
- create a lookup table to display the component list of the household;
- call the Labour datamodel with the person id chosen from the component list;
- pass information from EntryDemRel to the selected Labour record;

- read the outcome code of the component after Labour is finished;
- update the outcome code in the ComponentList database.

Essentially, the Program Manager is the main part of the survey. It controls all the major datamodels of the survey except the Exit part. The CallSchedulerExit datamodel calls the Program Manager to start the interview. When EntryDemRel and Labour are done, the Program Manager is terminated, and it returns to CallSchedulerExit to start the Exit component.

Controlling the Program Manager

Since the application begins with CallSchedulerExit, we need a way to call the Program Manager from CallSchedulerExit. We can simply use a function key to execute an external program. However, it is difficult to control the sequence of events. We only want the Program Manager to be executed once at the beginning of the application. By using the function key, it can be executed whenever the function key is pressed, which is not what we want. Also, we have to pass certain information from CallSchedulerExit to the Program Manager, which cannot be done with the function key. Moreover, the clients do not like the idea of pressing a function key to start EntryDemRel. They would like to see the application to continue with the normal flow of the questions. Function keys should be used for specific tasks that deviate from the application, instead of continuing with the next part of the application.

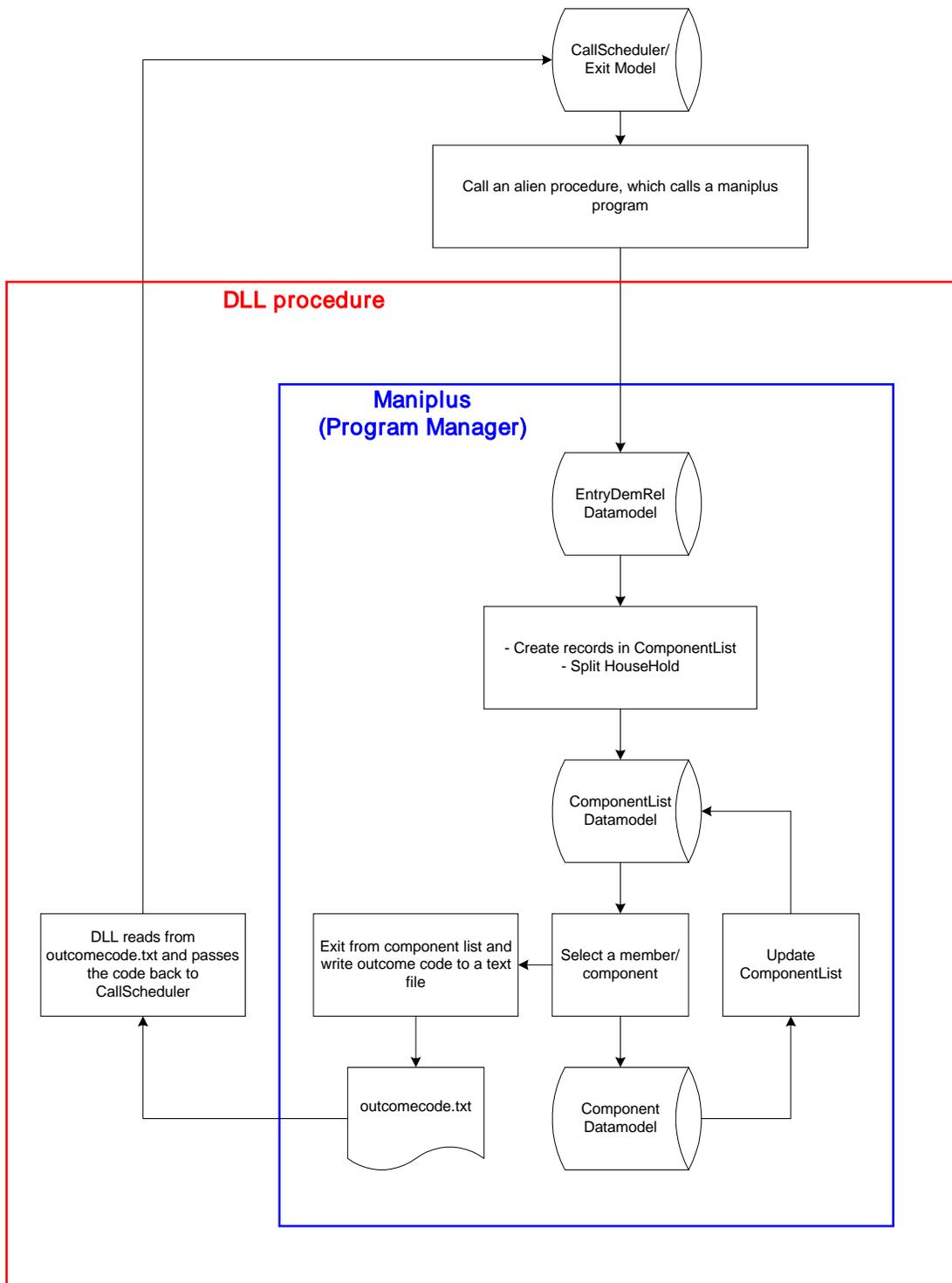
The other way to call the Program Manager is to use a DLL procedure. It is easier to control a DLL with an alien procedure in the application. We can simply set a flag in the application to make sure the procedure will only be executed once. Responding to a specific question in the application can trigger the alien procedure, appearing like we are following the normal flow of the questions, rather than starting a new process. We can also pass any fields in CallSchedulerExit to the alien procedure as parameters, so that they can be used in the DLL procedure and the Program Manager. The DLL procedure also allows us to change the values of the parameters, i.e. updating those fields in CallSchedulerExit. Updating a field of the current form with an external program is not possible because the form is locked by the DEP. The only way to do it is to use a DLL procedure. The field that we need to update is the outcome code, which can be assigned automatically based on the outcome codes of the components of that case. We can update other fields the same way for other surveys in the future. For now, we will only update the outcome code.

How does it work

The DLL procedure is not complicated. Its main function is to call the Program Manager with some parameters. For SLID, we are passing household id, outcome code, interviewer language, and respondent language as the parameters. In CallSchedulerExit, filling in the first field will call the alien procedure that activates the DLL procedure. The DLL procedure uses the parameters to call the Maniplus program, Program Manager. Program Manager uses the household id to call the corresponding case in EntryDemRel. Using the interviewer language and the respondent language, the corresponding question texts and menu will be used to open the case in EntryDemRel. The DEP for EntryDemRel is then opened for data entry. When EntryDemRel is finished, it returns control to the Maniplus program. Using the household id as the primary key, the Maniplus program reads from EntryDemRel. After reading the case from EntryDemRel, the Maniplus program has to do two things. First, it determines if anyone moves away from the household by checking the roster information in EntryDemRel. If someone moves away, it splits the household by adding a new case in CallSchedulerExit and EntryDemRel in order to create a new household. Second, it checks the demographic information to see who is eligible for the Labour component, and then updates the ComponentList database with the person ids of all the candidates. A lookup table window, ComponentList, showing all the candidates in that household will then be displayed. The interviewer picks a member from the lookup table, and the Maniplus program uses the person id as the primary key to call Labour. The language parameters from the DLL procedure will allow the DEP to use the proper menu and language texts. After Labour is finished, the Maniplus program

regains control and reads the Labour database to get the outcome code of the Labour component. It updates the outcome code in the ComponentList database, and displays ComponentList with updated information. The same process is repeated until the interviewer exits ComponentList. At this point, the Maniplus program uses the household id as the secondary key to search for all component outcome codes for that household in the ComponentList database. The outcome code for the household, if applicable, is determined by the combination of all the component outcome codes. The household outcome code is written to a text file, and the Maniplus program is terminated. The DLL procedure regains control and reads from the text file to get the household outcome code. It updates the outcome code parameter and returns to CallSchedulerExit. After it returns to CallSchedulerExit, it goes to the Exit block and asks the proper questions to finish the case. When the case is closed, the callscheduler delivers another case, and repeats the whole process. The diagram below (figure 1) shows the sequence of events for the process.

Figure 1. SLID (Jan 2001)



Other parts of the application

The Program Manager is the core part of the application that collects data for the survey. However, we also need other functions to support and complete the application. For example, being able to display demographic information, and call summary at anytime during the interview. Also, the ability to view and edit notes at anytime. These functions can be implemented using Maniplus. For demographic information, we use a Maniplus program to read from EntryDemRel and display the information in a lookup table. For call summary, we read the CatiMana block from CallSchedulerExit and display the information in a lookup table. We use a separate Blaise database to store the notes. Since we have to be able to edit as well as view the notes, it is easier to use a separate database. If we put the notes on one of the main datamodels, we will be able to view the notes of the current case with a Maniplus program, but we will not be able to edit and save the notes because the DEP is locking the current case. So we use a separate database to implement the notes, and use a Maniplus program to access the notes database. These three Maniplus programs are assigned to three different function keys, F3 for demographic, F8 for call summary, and F11 for notes. The household id is passed as parameter when calling the Maniplus program. It uses the household id as the primary key to access the information for that household. Unlike the Program Manager, we do not have to call these three programs using a DLL procedure. We want to be able to call these programs whenever we want, and we do not have to update any information when we return from the programs to the main application. Therefore, it is not necessary to use a DLL procedure to control the Maniplus programs. We can simply use the function keys to call the programs directly.

There are two other databases that are important to the application. One is the Users database and the other is the Routing database. The Users database stores the id, the main group and the other groups for each interviewer. The Routing database stores the routing rules for each outcome code. When an interviewer from a certain group exits a case with a certain outcome code, that case could be routed to another group of interviewers, or routed back to home if it is finalized. For example, if an English case is assigned the outcome code that indicates the respondent prefers the other official language, that case will be routed to the French interviewers. Routing is accomplished by changing the ToWhom field. If a case has to be routed to a specific group, the ToWhom field will be changed to the appropriate group name. If a case has been finalized, the field will be changed to "Home". Each routing rule consists of the survey id, the original group, the status, the outcome code, and the destination group. The application uses the survey id, the original group, the status, and the outcome as the primary key to search the Routing database and retrieve the destination group. It uses the destination group to update the ToWhom field. Each case is assigned to a certain group, which is indicated by the ToWhom field. Changing the ToWhom field makes the case accessible to the proper group of interviewers. In the application, CallSchedulerExit reads the external database, Users, to get the main group of the interviewer. After an outcome code has been assigned to the case, it reads the external database, Routing, to get the destination group for the ToWhom field. The ToWhom field of the case is used as the original group when searching the Routing database. However, that field may contain the interviewer id instead of the interviewer group, if the manager assigns the case to a specific interviewer using the CATI Management program. This is why we have to read the Users database to get the main group of the interviewer. If the ToWhom field is the interviewer id, there will be no match in the Routing database. So we use the interviewer main group as the original group to do the search instead.

The last part of the application is the UpdateDaybatch program. It is a Maniplus program that puts certain cases back to the daybatch. Normally, the Blaise callscheduler will put a case back to the daybatch if it is Busy, No Answer, Answering Service, or Appointment. However, many of our cases do not fall into those four categories. Cases such as tracing required, language barrier, refusal, request for interview by another interviewer, etc. They should all return to the daybatch in the same day with default priority. Assigning these cases to one of the four categories will not be appropriate because those treatments will assign certain priority to a case. We want those cases to come back as if they were untouched cases. Since currently there is no treatment in Blaise to do what we want, we have to create our own solution. In

the application, we treat all the special cases as Others. The UpdateDaybatch program reads the CallSchedulerExit database to check if a case has been treated as Others. If the last dial result is Others, and the ToWhom field is not "Home", that means the case has not been finalized and should be returned to the daybatch. The program uses the daybatch_add function to add the Others cases back to the current daybatch. By running the program in a regular interval manually or automatically, the daybatch is being updated constantly. All the cases that have not been finalized would be accessible through the Blaise callscheduler.

The Results

After almost a year of analysis, design, implementation and testing, the SLID application was made available January 2001. The application was deployed to all seven regional offices, and production began with the first full Blaise CATI application. Like any new applications, SLID did encounter some problems.

One was a delay between CallSchedulerExit and Program Manager. When CallSchedulerExit activated the DLL procedure to call Program Manager, there was a delay about 10 to 30 seconds. The reason for the delay was the initialization of the Maniplus program. Program Manager controlled the communication between all the datamodels, so we had to put them all in the Uses section of the program. Every time Program Manager was called, it had to load the meta-info of all the datamodels, thereby creating the delay. The duration of delay increased as the network traffic increased. We tried to use the filter setting to filter out part of the datamodel, but there was no obvious improvement. We were told that if a Maniplus program uses all the datamodels, and it calls a second Maniplus program that uses the same datamodels, the process of initialization of the second Maniplus program would be much shorter. However, it did not work in our case. We used a Maniplus program to call CallScdehulerExit, but that same program did not call Program Manager. It was called by a DLL procedure within CallSchedulerExit. Therefore, we had two separate Manipula processes running. Opening all the datamodels in the first Maniplus program did not benefit the second one in this case. Fortunately, the interviewers were supposed to phone the respondents after Program Manager started EntryDemRel, so there was no interruption during the interviews. However, it is still a technical problem remaining to be solved.

Another problem was in CallSchedulerExit. After the application returned from Program Manager to CallSchedulerExit, we started to ask the Exit questions. Sometimes there was a few seconds delay between questions in the Exit block. Later we discovered that the reason for the delay was the external read to the Users and the Routing database. In the early stages of development, the watch window had not been introduced as a debugging tool, so there was no way to tell when and how the external database was read. After the watch window became available, we obtained a better understanding of how the external database was accessed. We improved the situation by changing the way that we did the external read. The delay remained however not as serious as before. We suspect that there is a pending search for any unsuccessful search on the external database. Basically, the application would not stop the external read until it successful read something from the database. This is only our theory, and we are still investigating this issue.

There were other problems that were not related to the design or the implementation of the application. Sometimes the application froze on one workstation, and the whole system would start to slow down. Eventually, other workstations seemed to freeze as well. When this happened, we found multiple entries of the same case in the history file (.BTH file). These entries had the same start time but different end time. We located the workstation that was accessing the case at that time. After we rebooted that workstation, the system returned to normal and the rest of the workstations were unfrozen and working again. We concluded that it was not an application problem but a Blaise problem, as the problem

surfaced in two other applications that went into production after SLID. The two applications were very different from SLID, i.e. they were single datamodel applications, and they were less complex than SLID.

Also, there was problem with the Blaise callscheduler. It was looping within a certain part of the daybatch. As a result, it ignored most of the untouched cases, and kept delivering previously attempted cases to the interviewers. To solve this problem, we had to explicitly exclude all previously attempted cases from the daybatch for a few days in order to get to those untouched cases. We reported this problem to Westat and Statistics Netherlands. They confirmed our finding, and fixed this problem in the subsequent release of Blaise.

Since the Blaise callscheduler was not delivering cases properly, the interviewers were trying to use the databrowser to get cases, which created another problem. Whenever several interviewers tried to use the databrowser at the same time, everything started to slow down. It took a very long time for the databrowser to scroll up or down, and the interviewers experienced long delays within the application. This did not happen when the databrowser was not used. Since the default databrowser was constantly connected to the database, it created a lot of network traffic. The more the databrowser was used, the worse the problem became. To solve this problem, we had to implement our own databrowser using Maniplus. The Maniplus databrowser program required the interviewers to search on the primary or secondary keys to limit the number of cases to be displayed. It read from the database in shared mode, at the rate of about 2 cases per second. This databrowser might not be fast while retrieving cases, but it did not slow down the system afterward as it was no longer connected to the database. It gave the interviewers an instant view of the database, and also provided other valuable functions for the interviewers. Before, the interviewers had to actually get into a case in order to use the function keys F3, F8, and F11 to view demographic, call summary, and notes, which was quite time consuming. In our browser, we included three control buttons that activate the Maniplus programs for those functions. The interviewers could pick a case from the browser, and they could then decide whether to access it or just look at the information. Not only did we solve the databrowser problem, we even enhanced the application and created a prototype browser for future use.

Conclusions

SLID 2001 was a success despite the problems we encountered. It showed that our new design is feasible and sound. We now need to focus on improving the design or the way that we implement the application, so that we can eliminate the existing problems. The major concern that we have is the delay between CallSchedulerExit and EntryDemRel. We know that the delay is the result of the long initialization of the Maniplus program. So the question is, how can we shorten that time? There are many things that could contribute to the delay, such as the number of datamodels, the size of datamodels, the number of interviewers, traffic on the network, and the way that we open the datamodels. Although this delay does not interrupt an interview, it will eventually affect productivity, so we must shorten the delay to an acceptable level. Basically, there is no delay within each individual datamodel. The only datamodel that has occasional delay is CallSchedulerExit. We know that this delay is probably caused by a constant external read. If we control the external read properly, we will likely eliminate this problem.

There are other improvements which can be made if additional functionality is made available in Blaise. For example, if Blaise can introduce a treatment that returns a case to the daybatch with default priority, we would not have to use our UpdateDayBatch program. The concept of the UpdateDayBatch program works, however we have to schedule it to run regularly. When it runs, it has to access the database and update it, thereby competing with the interviewers, which may affect the performance of the application. Almost every social survey has the same kind of cases that need to be returned to the daybatch. Without the proposed treatment, we will need to prepare an UpdateDayBatch program for each and every survey. This makes it hard to maintain the applications and the standards.

The efficiency of the Blaise databrowser is another thing that requires improvement. Although the databrowser we developed is an ideal replacement for the default one, we still have to develop one for each survey. For some small surveys that do not require the full functionality of our customized browser, it may be easier to use the default one if it does not slow down the system. Nevertheless, in the process of handling the browser problem, we developed our own browser with customized functionality, which clients have found very useful. In fact, it is so useful that we are going to make it a standard feature for all new surveys. Our solution to the problem actually becomes the prototype and the blueprint for the next customized browser.

The experience with SLID 2001 reinforced the potential of the multi-datamodel design, and will enable us to further improve our upcoming surveys. These surveys will also benefit from functions like UpdateDayBatch and the customized browser. SLID 2001 also highlighted some limitations and problems in Blaise and Manipula. The reporting of these problems, as well as their subsequent correction in new releases of Blaise, is of benefit to all Blaise users. With Blaise being constantly enhanced and with our growing expertise with the product, we will continue to see new and improved Blaise CATI application in future. SLID was just the beginning for us.

Session 4: Case Management II

- A Windows-Based User Interface and CAPI Information System
Vesa Kuusela, Statistics Finland
- Redesigning CAI Systems to Handle Multiple Authoring Languages
Michael Haas, U. S. Bureau of the Census
- The Annual Survey of Manufactures (ASM) at Statistics Canada
Okuka Lussamaki, Statistics Canada
- Case Management for Blaise Using Lotus Notes
Fred Wensing, Australian Bureau of Statistics
Brett Martin, Statistics New Zealand
- LWR: An Integrated Software System for the Household Expenditure Survey
Thomas Pricking,
Federal State Bureau of Data Processing and Statistics, Germany

A Windows Based User Interface and CAPI Information System

Vesa Kuusela, Statistics Finland

1. Introduction

In the first quarter of year 2001, the whole CAPI information system at Statistics Finland was changed from a DOS based system to a completely Windows based system, including the latest Blaise 4 Windows (B4W) version. The change was put in effect at the same time when interviewers received new laptops. Up to that point, the part of the system that resided in interviewers' laptop was still entirely DOS based. The office software was largely Windows based already but it was in 16 bit architecture and therefore it was rewritten.

The new operating system and graphical user interface provide many new facilities, which did not exist in a DOS based system. Actually, the graphical user interface and the fact that the screen size of the interviewers' new laptops increased considerably (1024*780 pixels) led to new design of the interviewer interface. The basic structure of the interface was tried to keep as close to the previous one as possible, to keep learning to use the new system easy. However, from the very beginning it became obvious that the layout of the interface had to be designed on a different basis. Also in a smaller screen (800*600 pixels) the previous layout had been inappropriate. Though the layout of the interface had to be redesigned, its functionality was retained.

On the other hand, there was no need to introduce major changes the basic architecture and basic principles of the previous system, even though the change in the visible part of the system was considerable. The infrastructure of the CAPI information system was designed and installed in mid 90's and it has been functioning well. The infrastructure and design principles were described at IBUC 1995 (see Kuusela, 1995) and IBUC 1997 (see Kuusela and Parviainen, 1997). Basic idea has been so-called object-based design which has proved to be reliable and flexible. Visual Basic was used as the software tool, as in earlier versions. Now only Visual Basic version 6 was used, instead of three different versions like before.

Modern programming tools enable a lot of new solutions and provide many new features. On the other hand, programming in Windows environment is full of pitfalls and the system needed much more testing than in the DOS era. Introducing

modern new features makes the system more complicated and by that more vulnerable. Design and programming proved to be much more demanding than before. Especially the use of concurrent forms brought up situations which compelled to test carefully the system.

This paper describes the system and interfaces as they are now and tries to explain why certain decisions were made.

2. Basic structure of a CAPI system

From the users' point of view, a CAPI information system may be seen to be composed of two user interfaces¹, interviewers' interface and supervisor's interface, and a communication channel between them (see figure 1). Usually the communication channel is composed of a modem connection and telecommunications software but it could be done with other methods, as well.

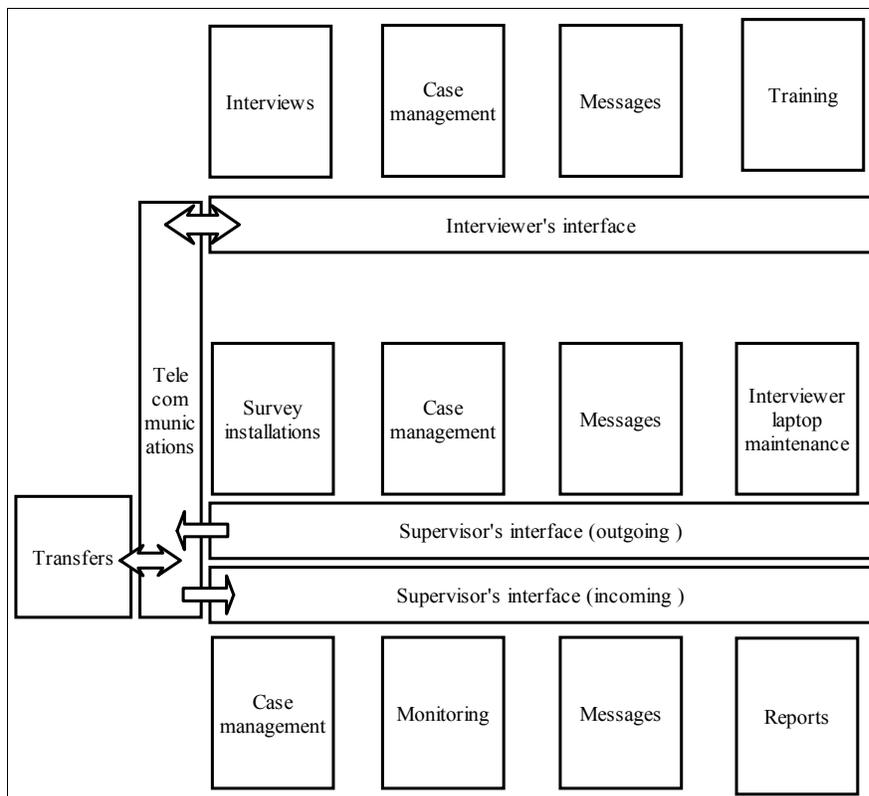


Figure 1: A schematic display of user interfaces of a CAPI system.

Several different tasks are embedded in both interfaces. Most of the tasks have a counterpart in the other interface and a task should be able to communicate with the corresponding task in the other interface. For instance, a message from supervisor should end in an interviewer's message box in a readable format.

The main function of the interviewer's user interface is to support

¹ An interface, or an user interface, is a piece of software that connects users to an information system, like a dashboard.

interviewing and to take care of the local case management tasks. Ideally the case management should be a transparent part of a transparent interviewer interface, which should facilitate the interviewer's work (rather than make it more difficult). An objective of the design should be to free interviewers to concentrate on their actual work.

Supervisor's interface has two parts: for outgoing activities, like installation of new surveys; and for incoming activities, like assembling a single data file from interviewer files. In fact, there are several important, even critical, functions in the supervisor's interface, besides case management, because whole information system is controlled by that.

The user interfaces are connected to the information system, which also has two different but equally important components. On the other hand, the information system has only one infrastructure that makes it function as expected, and that keeps it together as an integrated whole. Case management is the major task of the information system but there are also many different tasks, which the information system is supposed to handle (see also Nicholls and Kindell, 1993).

2.1. Interfaces

Interfaces, in general, should be designed according to different standards to different types of users, especially if they have to carry out different tasks, or if they have different capabilities to use computers. A key concept in the design of user interfaces is usability. A lot of research has been done on this area, but also usability testing is needed in each specific application. Typically, the interface which is designed for interviewers should be well thought out because interviewers are selected to manage as interviewers, not as computer specialists, and the techniques should help them. On the other hand, one can expect much more computer skills from the personnel in a statistical office.

Interviewer interface

The first idea was to keep basic structures of the interfaces as close to the previous interface as reasonable, but the graphical user interface, large screen size of the laptops and the possibility to use the mouse compelled to redesign of the interviewer interface. Only the functionality was tried to keep similar as it was. Supervisors' interfaces, on the other hand, did not change much because they were Windows based already.

In the new interface version an interviewer is able to manage practically all situations by two different displays: one where the descriptions, status and samples of the active surveys are shown (see figure 2); and one where is displayed all

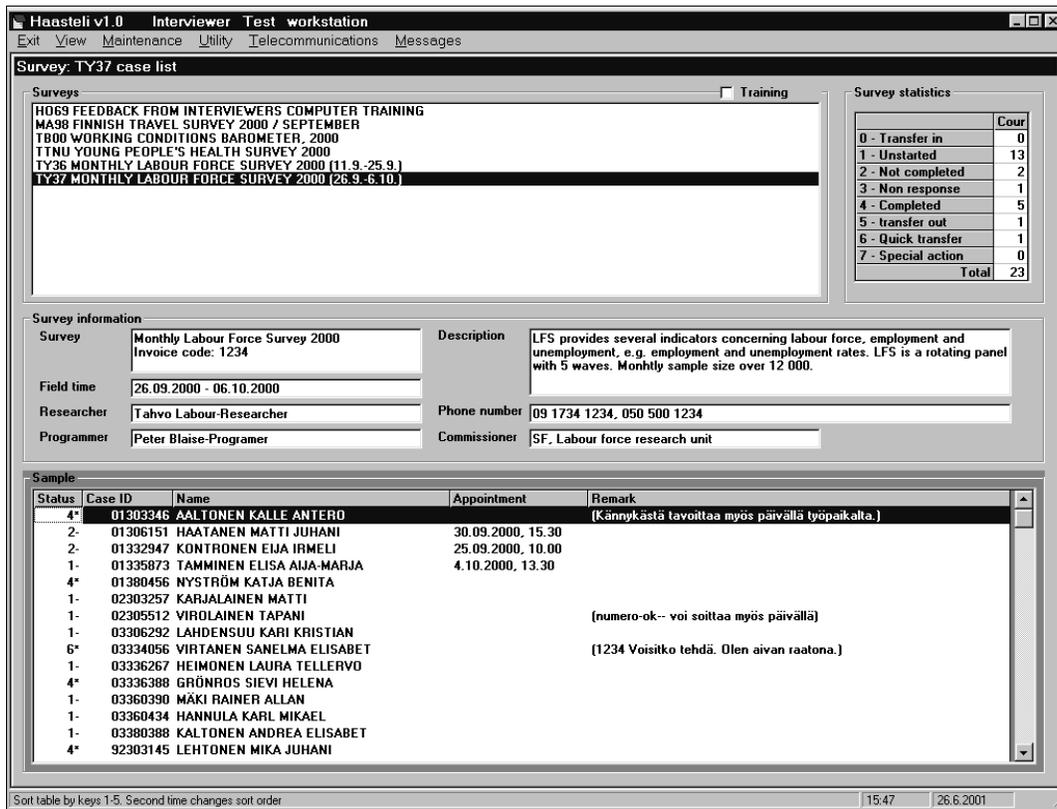


Figure 2: The main view of interviewer's interface. At the upper left corner are the active surveys and beside them is statistics of the selected survey. Below is the sample of the selected survey and in the middle some information of it

available information about the selected case (see figure 3). All operations can be done either by the mouse or by the keyboard.

The main view of the interfaces can be changed to two other views: in one view, only the cases of a selected survey are displayed without other information of the survey; in the other view, all cases of all surveys are displayed. In all the three different views, the case list

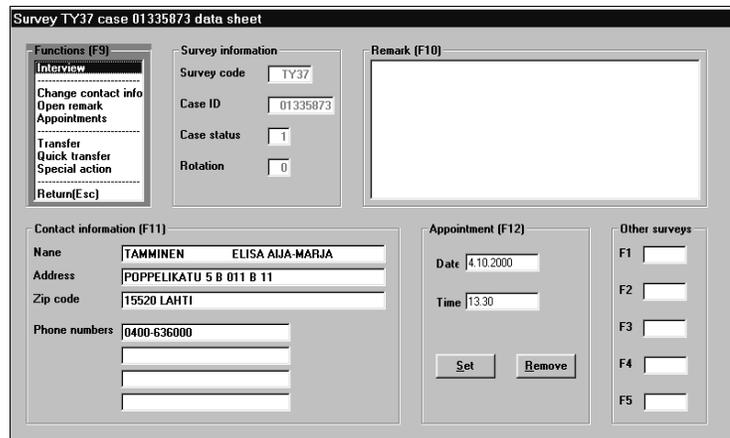


Figure 3: Data sheet of a selected case.

may be sorted according to all fields in it (i.e., status, ID, name, appointment etc.).

All the functions which an interviewer can apply to a case are shown in the case data sheet (see figure 3), including appointment and remarks. Interviews are started here and after an interview control returns here and the status of the case has to be updated before the interviewer can continue. After that, the completed cases are moved automatically in the out queue.

In designing a large screen a new problem emerged: how much and what kind of information is reasonable to place in one display. Much more information could have been placed in each screen but the result had been incoherent and tangled. The screens shown here are compromises.

Blaise screen design and display standards.

The default screen layout of B4W, and the large screen of the laptops, made also the Blaise screen design necessary. In addition the facts that there were much more possibilities to design the screen layout in Windows Blaise than in Blaise III supported this. The default screen layout in Blaise 4 Windows is applicable in 800*600 screens but not in 1024*780 screens, because the font is too small and the horizontal division makes two wide and low sub screens which make question text hard to read. We must bear in mind that illumination conditions between and during interviews may vary considerably - and interviewers are mainly middle aged.

The wealth of design possibilities in Blaise brought up the need of standardization of the screen layouts, in order the questionnaires to have similar appearance in all surveys, and also to give support to standardized interviewing. If every author was allowed to select the colors and fonts and maybe even the general layout of the screen freely, questionnaires would probably become very different which in turn would be very confusing and detracting for the interviewers.

However, the standardization of questionnaire screen layouts appeared to be more difficult a task than anticipated, and final standard has not been set up to this point. Some alternative standards have been tested but all of them have encountered some criticism. An agreement has been achieved only about the font (Lucida sans unicode, 11 points), and the about dividing the screen vertically. Obviously, some research and some planing and a lot of usability testing will be required before an optimal and ergonomically effective layout standard can be reached.

Supervisor Interfaces

Supervisors have many different tasks and they need several interfaces to take care of their job. For instance in Statistics Finland there are more than ten different interfaces in daily use. In figure 4 is shown one, so-called Survey Starter. This is needed when a survey is installed on the field. Actually Survey Starter is a collection of buttons which launch other programs and the arrows indicate in which order one should proceed. The results of Survey Starter are packets for interviewers including the questionnaire, the sample, background information for the sample and all the necessary tools for the interviewer interface to do the installation. Each interviewer will get her packet during the next telecommunications session.

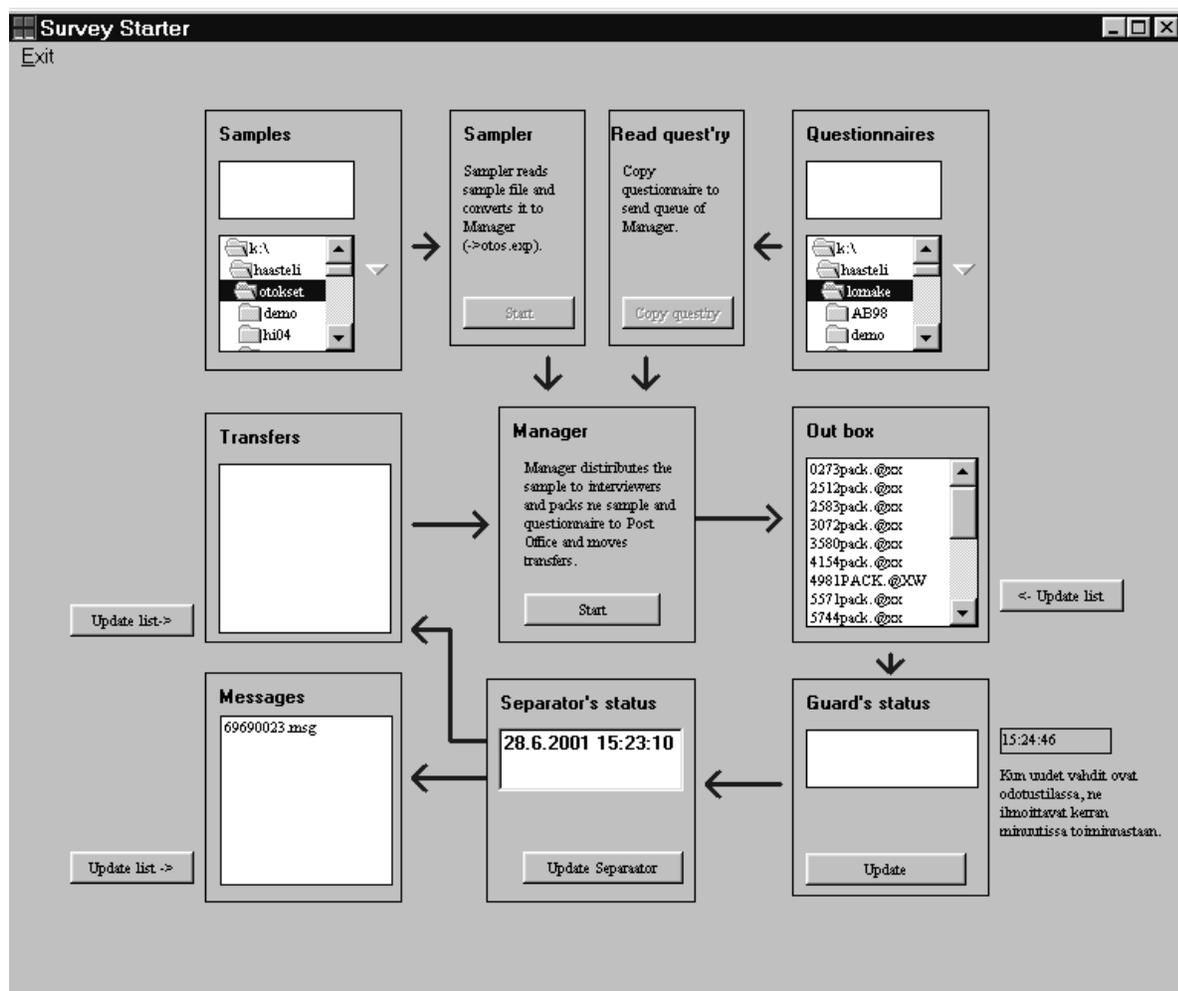


Figure 4: The supervisor's interface by which a survey is installed. Actually this is a collection of buttons which launch other programs and the output are the packets for interviewers in the out box.

2.2. Information system

Apart from interviewing, the software in interviewer laptop has to perform several critical tasks, e.g., installation and maintenance of surveys, telecommunications, case management, some laptop maintenance. Most of the tasks should be automatic so that they do not require interviewers to intervene in the processing. If the operations would require complicated computer tasks from interviewers, it would increase substantially the liability on malfunctions in the system. And to be able to concentrate on their main duty interviewers should not be loaded with additional tasks on which they not trained for. To achieve this goal the interviewer interface was designed as a platform where surveys are installed and which is partly operated by supervisors.

Case management

Case management in a CAPI system means the logic of the flow how the sample points, cases, go through the system. That involves several tasks: exclusive and exhaustive distribution of the sample to selected interviewers, delivery of the sample to interviewers; forwarding the case to the CAI software to be interviewed; maintenance of the status of each case; management of the case transfers from one interviewer to another in all stages of the survey; gathering of cases back to office; assembling one data file from the slivered sample; enable the monitoring of field work and reporting.

Roughly speaking there are two major approaches to design the case management for a CAPI system: a data base -based information system and an object-based system.

In the data base -based approach a file, a part of the master database, containing all sample points for a particular interviewer is sent to his/her laptop. A data base -based system involves some difficulties and dangers which need careful design: e.g., maintenance of the integrity of the database; how many interviews are lost if the database is corrupted; maintenance of the status of the cases; at what point and how the cases are sent back to the office.

In the object-based system, each sample point constitutes a single object. That is, all the necessary information to conduct an interview is encapsulated in one file. An object may contain an unique identification, status, sample data to contact the interviewee or household, additional data of the sample point to be used in the

questionnaire, messages, etc. When the object is returned, it includes updated data (e.g., status) and the answers to the questionnaire attached.

The object-based approach provides many assets compared to data base -based approach (see Gray, 1995 and also Rumbaugh, et. al, 1991). For instance, the assignment of cases to interviewers becomes fairly straightforward and there is no danger that the same sample point goes to two interviewers; and case transfers from one interviewer to another may be handled reliably. If one data file is corrupted only one case is lost in the object-based system.

Once the object-based approach is adopted, it also makes the activities of CAI system, rather straightforward. Moreover, in the office system, each case may be processed separately in a so-called 'flow basis' (see Gray and Anderson, 1996) almost immediately it has arrived and hence interviews and office editing may overlap.

Already in the previous version of the CAPI information system of Statistics Finland object-based architecture had been adopted. There was no need to change it, especially because some changes in Blaise have facilitated its usage. Technically the infrastructure of the information system follows loosely the ISO-OSI network protocol where the communicating parts are CAI server and the interviewer workstation although they are not hierarchically ordered. The mental image in the design of the data flow was the operation of post offices and mail delivery. For instance, in the work station there are boxes (dictionaries) for incoming and outgoing mail, and in the CAI server there are mailboxes for each interviewer and survey.

Telecommunications

Completely new software was designed for telecommunications. Also here the functionality was kept close to previous one but the result was quite different. Basically, there is now only one telecommunications program that has two instances communicating with each other: one in interviewers' laptops and one in a telecommunication server (called Guard). The part running on a telecommunication server is like a postman getting packets from interviewer works station and delivering packets from interviewer's mail box (in CAI server) to a workstation. The part running on interviewer's workstation is passive only performing commands coming from the server.

In the telecommunications server also another program is running constantly: so-called Separator unzips and delivers incoming objects in correct folders. There are several different objects in the system (and new objects can be defined without difficulty), which can be discerned by the extension of the file name. Therefore, Separator does not need to open the objects.

Naming convention

One of the corner stones of the system is the naming of the files (i.e., objects). The file name of the sample point indicates both the interviewer and the respondent. The extension of the file name indicates the type of all objects.

The naming convention has two objectives: first the appropriate handling rules and methods can be applied to the objects; and secondly the basic statistics concerning field work can be produced from the file names without opening the files

Several different objects have been defined in the system. For instance, one object type is a batch file, which contains commands which are known by operating system. The interviewer interface reacts immediately if it notices a batch object and launches it. With the batch object, greatest part of laptop maintenance can be taken care of.

3. Other development

Interviewers' laptops were furnished also with Internet access, and each interviewer was provided also with an email address. The rationale for the Internet access was that telephone numbers and street addresses may be obtained via Internet. In the future, Internet will be utilized much more, for instance interviewers can get instructions and survey results from the web. There are many different plans on where Internet and email could be used. However, before anything decisive can be done, all interviewers must have adequate Internet skills. In the beginning, some 25% of interviewers had great difficulties.

The message system embedded in the CAPI information system and the ordinary email do not replace each other. The old message system is needed to inform interviewers about issues closely related to data collection. The ordinary email will be used in less critical information.

4. Discussion

The functionality of B4W is close to that of Blaise III and therefore interviewers could learn it easily. Also the functionality of the new user interface was reasonably close to the previous one and therefore easy to learn. The most difficult part for some interviewers was the new Windows environment, that is, a graphical user interface and the use of the mouse. Many of those interviewers who had not used Windows before, experienced great difficulty using the mouse both in pointing an item and clicking mouse buttons (see also Chou and Torn, 2000).

The new operating system and graphical user interface brought up many new facilities which did not exist in DOS based systems. However, the recent advances of computers and in software technology do not affect much the classical ADP problems like data structures and system architecture. The major changes can be seen on screen designs (which actually did not exist previously) and in the problems of programming which are related to the graphical and concurrent window type user interface. The system becomes easily very complicated and vulnerable. Design and programming can be very demanding.

In some respects, the change from DOS based system to Windows based system appeared to be as great a change as it was to start with the DOS based system eight years earlier.

In Windows environment the usability issues are much more difficult to solve than they were in DOS environment, due to the vast number of alternative features and their combinations. Also, the use of concurrent forms on a single display may be difficult to handle. Usability of the interviewer interface needs special attention especially because all interviewers are not well in with Windows environment. In a CAPI system, usability is a major issue, as emphasized also by Bushnell (2000) and Hansen et.al. (1997).

According to one definition, usability means that people who use the software (and the computer) are able to work productively to achieve their goals without difficulty, in their own physical and social environment. In other words, usability should produce such additional value that the users notice it. Achieving this in a CAPI system running in Windows environment is a challenge.

References

- Bushnell D.:** From Dos to Windows: Usability issues for Interviewers. *Presented at 6th International Blaise Users Conference, Kinsale, Ireland, 2000.*
- Hansen S.E., Fuchs M., Couper M.P.:** CAI Instrument Usability Testing. *Presented at the annual of the American Association of Public Opinion Research, Norfolk, USA, 1997.*
- Gray J.:** An Object Based Approach for the Handling of Survey Data. In Kuusela V. (ed.): *Essays on Blaise 1995.* Statistics Finland, 1995.
- Gray J., Anderson S.:** The Data Pipeline - Processing Survey Data on a Flow Basis. In Banks R., Fairgrieve J., Gerrard L., Orchard T., Payne C., Westlake A. (eds.): *Survey and Statistical Computing, 1996.* Association for Survey Computing, 1996.
- Kuusela V.** Interviewer Interface of the CAPI-system of Statistics Finland. In Kuusela V. (ed.): *Essays on Blaise 1995.* Statistics Finland, 1995.
- Kuusela V., Parviainen,a.** An Object-Oriented Case Management System for CAPI Surveys. *Actes de la 4^e Conférences Internationale des Utilisateurs de Blaise.* INSEE, 1997.
- Nicholls W.L., Kindell K.K.:** Case management and communications for Computer Assisted Interviewing. *Journal of Official Statistics.* 1993, 9, pp. 623-639.
- Rumbaum J., Blaha M., Premerlani W., Eddy F., Lorensen W.:** Object-Oriented Modelling and Design. Prentice Hall, 1991
- Schou R, Dorn T.:** NASS Conversion to Blaise 4 Windows with a Visual Basic Interface. *Presented at 6th International Blaise Users Conference, Kinsale, Ireland, 2000.*

Redesigning CAI systems to handle multiple authoring languages

Mike Haas, US Bureau of the Census

Background

After a period of evaluation, the Census Bureau made a decision to use the Blaise system to develop the questionnaire for the Consumer Expenditure Quarterly survey. Until that time, the primary data collection tool for most of our demographic Computer Assisted Personal Interviewing (CAPI) and Computer Assisted Telephone Interviewing (CATI) surveys was the Computer-Assisted Survey Execution System (CASES) software developed by UC Berkeley. The decision to use Blaise made a significant impact on our CAI control systems. Not only were we in the process of converting our control systems at the data collection level from DOS based to Windows based, but we would also now be administering two different data collection software packages. A decision had to be made whether to build a new system for Blaise surveys or to re-engineer our current system to handle both authoring languages (and be able to accommodate other packages that might be desirable in the future). We decided on the latter approach.

The Consumer Expenditure Quarterly (CEQ) survey was not the first experience the Census Bureau had using Blaise because the Failed Edit Follow Up of the American Community Survey (ACS) was authored in Blaise. A customized control system was constructed with the assistance of Mark Pierzchala of Westat for the collection of the ACS Failed Follow Up data. The CAPI and CATI portions of the ACS were collected with a CASES instrument and administered under our standard control systems.

The CEQ survey would be the first Blaise survey to be integrated into our standard collection and control systems and thus required the re-engineering of those systems. Once it was determined that we would now be administering more than one type of authoring language, we had to rethink the process.

Processing systems in the data collection process.

There are several systems that communicate with one another during the data collection process. This discussion will focus primarily on the process for demographic surveys as the majority of the field work is conducted for demographic surveys, although the process for Economic surveys is similar. The systems consist of

1. The Sample Control System (SCS)
2. The sponsor's survey specific processing system
3. The Master Control System (MCS)
4. The Regional Office Survey Control (ROSCO) system
5. The CAPI Field Representatives Laptop System (FRLS)
6. The CenCATI system.

See Figure 1 following.

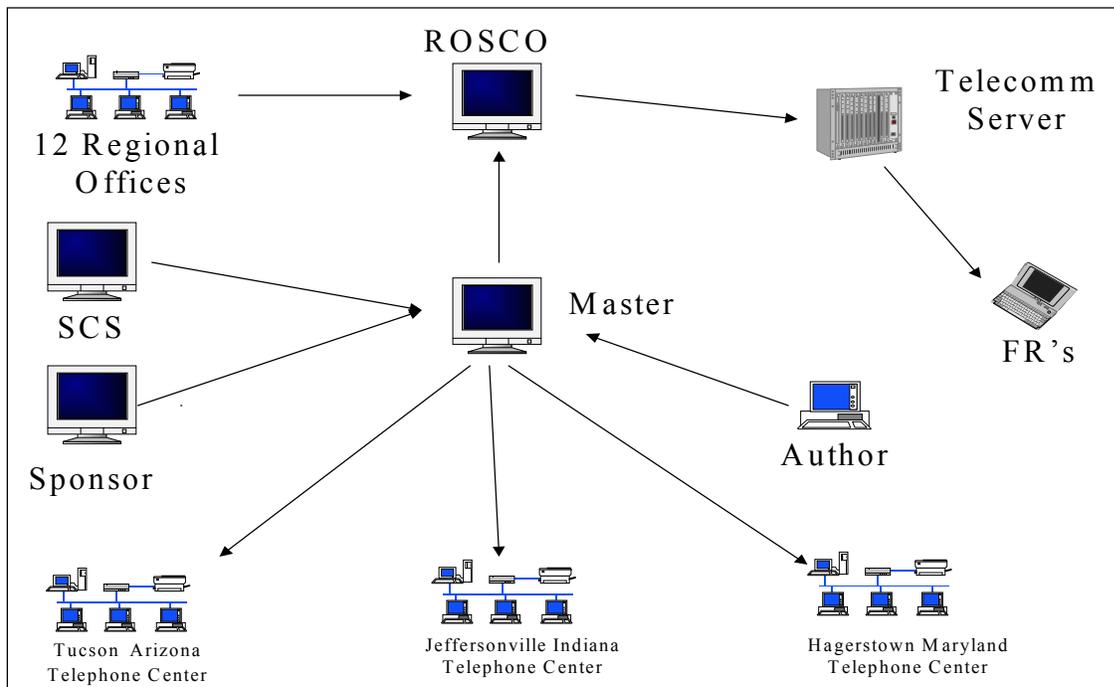


Figure 1: Tracking and Control Systems

A general synopsis of the flow follows:

The Sample Control System (SCS) maintains the universe of addresses initially obtained from the decennial census that is regularly enhanced by coverage improvement operations conducted throughout the decade. The SCS selects the sample of addresses to be interviewed for each of the major demographic surveys for each collection period and provides the control information for each case to be interviewed. The SCS delivers the file containing the control information for the sample addresses to the in-house survey sponsor. The sponsor's system merges any dependent data collected from prior contacts with the sample unit and produces a **Sample Control Input File (SCIF)**. The SCIF is delivered to the **Master Control System (MCS)**. The MCS installs the sample cases and tracks their progress through the data collection operation. When completed cases are delivered back to the MCS, the MCS produces output of the collected data to the sponsor for post collection processing. These outputs either consist of ASCII data or proprietary data (Blaise or CASES) according to the sponsor's wishes.

The authors (Blaise and CASES instrument programmers) design the instruments according to survey sponsor's specifications. As the authors design the questionnaire, they also take into account the relationships between the instrument and the SCIF as well as between the instrument and the various control systems they share data with. Once the instrument has been programmed and tested, the author delivers the instrument to the MCS where it is merged with the SCIF, and the data is loaded into case level proprietary databases. The case level databases are stored in **Binary Large Object (BLOB)** fields of an ORACLE database located on a Solaris/UNIX platform along with some case level administrative fields used to control and track the cases throughout the data collection process. Once the data has been loaded into the Oracle tables, the MCS distributes the CAPI cases to the ROSCO system and assigns the CATI cases to the appropriate Telephone center. The Regional Offices use the ROSCO system to manage and distribute the CAPI workload to the **Field Representatives (FRs)**. The FRs use a dialup connection to connect to the telecommunication server to pick up work and deliver completed cases. The

ROSCO system picks up the completed cases from the telecommunication server, updates status fields for the RO's use, and sends them back to the MCS to be output to the sponsor. Our current CATI system resides on Novell Servers and all of the cases to be completed at each telephone center are copied to the server to be worked on. Completed cases are sent back to the MCS each night to be output to the sponsor. In our new strategy for a revised CATI system, we would remove the necessity of the Novell servers and the telephone interviewers in the CATI centers will connect directly to the MCS Oracle database to pick up a case to interview. The call scheduling and case assessment programs will reside on the MCS. When the call attempt is completed, the case will be put back in MCS database for assessment and ultimately output to the sponsor.

Since our current production system has been evolving over the last 5 – 6 years, external users to the system (those providing inputs and receiving outputs) have finally become comfortable with the automated processes they have developed to interface with the system. We established standard file naming conventions that would categorize the various input and output file types so that one could easily distinguish the contents of the file by the name given to it. The file name identifies the mode of collection, the survey name and collection period, the month and day the file was created. The extension assigned to the file identified the type of contents contained within. Each survey also had its own directory structure on the Master Control System where the sponsor's data processors could deliver inputs and pick up outputs automatically from their systems. They were provided with programs in DOS, Unix and VAX (depending on their processing platform) that they would execute to deliver inputs from their machine to Master Control and pull outputs from Master Control to their platform. These programs would also do some housecleaning on Master Control. Now that the sponsors had a standard way of interfacing with Master Control, we wanted to make sure that their interaction with the new system as seamless as possible. Likewise all of our internal processes that handle the CASES instrument data have been fairly well standardized for all surveys. We also wanted to retain as much of the coding for those processes as well.

Standardized Interfaces

Since many diverse questionnaire instruments flow through the system, it was critical that the interfaces between the instruments and the systems as well as among the systems be as standardized as possible. We learned this early on in our administering of CASES surveys and we continued this practice as we migrated to Blaise. In order to expedite the processing between the systems from survey to survey, standard file formats were established to transfer control data between the systems. Each of these standard formats that we refer to as "record types" contain homogenous data that all surveys use. One record type contains all of the fields the Master control system needs to track a case, another contains address information, another contains information the ROSCO system needs to make interviewer assignments, etc. These record types are built into all instruments, although not all surveys use all fields.

In surveys authored with the CASES software, standard .q files (somewhat comparable to either .bla or .inc files in Blaise) are created for each record type and included in each instruments. Likewise in Blaise, we have created 2 standard .inc files for each record type. One that defines all of the fields to be defined in the instrument's .bmi file and one that defines the ASCII data model for that record type that will be used in the manipula script to load the Sample Control Input File (SCIF). In addition to the standard record type input .inc files, we create a control.inc file that defines all of the other fields needed by one or more of the control systems that are not already included on one of the standard input record types of the SCIF. These standard record types and control fields ensure that field names, sizes and types will be compatible with the fields in the control systems. When authors design a new instrument they begin with these standard record types and the control.inc files as a starting point. All of the fields defined in the record-type and control.inc files are at the datamodel level. The purpose for this is to expedite the data

exchanges between the control systems and the instrument. The section titled “Manipula script generation for building interfaces” will provide a little more detail on this process.

Differences in handling Blaise and CASES Data

One of the major differences between the approaches of accessing proprietary data between CASES and Blaise is that CASES used utility programs that work the same for all instruments. There is no direct relationship between the utility program and the instrument or its data until runtime. For example to load the dependent data into the CASES proprietary database, one used the **setup** executable. The authors handle the link between the dependent data file and the instrument fields during instrument design. When the instrument was compiled, an **INDEX** file was created that identified the relationship between each field and its corresponding storage location as well as where each field’s data would be obtained from the dependent data file (just those fields that would receive input from the initial file). To load the data from the ASCII file into the CASES proprietary structure, one simply ran “**setup inputfile**” where inputfile contained the ASCII data to be processed. The **setup** program would then read the INDEX file and load the dependent data into the appropriate storage location. Another example is the process to produce ASCII data from the CASES proprietary data is accomplished by the CASES **output** program. By passing a file of field names and another file of caseids to the **output** program, **output** would generate an ASCII file containing the contents of those fields from the CASES data for the desired caseids. There was a fixed directory structure that CASES required to be in place for the software to function correctly, and the CASES utility programs acted on the instrument and the data that existed in the directory structure from which it was executed.

In Blaise, the same functionality is accomplished by preparing Manipula/Maniplus and Cameleon scripts. The programs that perform a similar process to what was done with CASES, are customized scripts that require the linking with the instrument data prior to runtime. These Manipula scripts need access to the compiled instrument during the prepare process. The instrument data model is highly integrated with the Manipula script and if there are structure changes made to the instrument, the manipula script that accesses it will most likely need to be re-prepared. Blaise allows a higher level of flexibility in accessing and formatting the output of ASCII data obtained from the proprietary data than CASES does, but also requires a little more customization. In CASES we could run the same program against all of our surveys without making any modifications, and therefore only delivered the program one time. In Blaise, since the Manipula scripts need to be compiled with the appropriate data model before they can be executed, survey specific versions are delivered to the client that will be executing the scripts along with the instrument. We did experiment with the **Get value** function of Blaise to see if we could pass in the name of the data model at runtime. This process worked but was very slow due to the size of the instrument and the number of fields being written out. It was much more efficient to compile the Manipula script for producing the output using the prepared instrument.

Another challenge we had to overcome was that most of our non-collection processing of CASES data was accomplished in UNIX. For Blaise surveys, we would try to accomplish these functions by rsh (remote shell) from our Solaris machine to an NT server and then run batch processes on the NT server to accomplish the tasks.

The above comparison was not to imply that the approach taken by Blaise or CASES was better than the other. It was simply to show the difference in the approach we would have to take into consideration to integrate the proprietary data with the control systems.

More Detailed overview of the instrument/system integration

After the survey sponsors provide the specifications to the instrument authors for the questionnaire and have determined the standard record types to be used for the specific survey in coordination with the Sample Control System (SCS), the SCS and the sponsors will merge the sample information with the dependent data from previous contacts and produce the Sample Control Input File (SCIF). The instrument authors pull the appropriate record type input .inc files for both the Blaise data model and the ASCII data models. They also pull the control.inc file. Then they program the instrument according to specs. In addition, the author writes a couple of Manipula scripts; one to load the SCIF data into the Blaise database and one that will be executed by laptop case management. The one executed by the laptop case management handles the interchange of data between case management and the instrument.

The author delivers the instrument and the manipula scripts to the Master Control System (MCS) and the SCS/Sponsor delivers the SCIF. The MCS pre-processes the SCIF by loading its database with some of the record types and updating some of the fields on the SCIF. One in particular is the assigning of unique case ID's to each case, which is used as the primary key for the case. The revised version of the SCIF is the one that will be processed by the manipula setup script. The MCS then runs a remote shell (rsh) to the NT machine containing the Blaise software. This shell executes the Manipula setup script to produce case level blaise databases. The name of the Blaise database files is the newly assigned case ID, (i.e. the primary key for the case). Each case's files are then zipped and stored into a BLOB field in the MCS database. The MCS determines the collection site where each case will be distributed and sets the site field accordingly.

For CAPI surveys, the ROSCO system reads the information from the MCS database and updates its databases accordingly. The ROSCO system will prepare the assignments for the Field Representatives (FRs) by pooling the appropriate Blaise data and the control information for the cases into a zip file, encrypting the file and placing it on the telecommunication server for interviewers to pick up. It also places the instrument and survey specific case management software on the telecommunication server for pick up by the FR's as well.

Included with the instrument are manipula scripts and a manipulus shell that handles the interface between the interviewer's case management system and the Blaise instrument. Since there are data fields that are updateable in both the instrument and case management system, the manipulus shell is a way to keep the common fields synchronized. The interviewer's case management system is a power builder application built around an Oracle Lite database. While the interviewer is viewing the cases in his/her assignment, a view of a number of the fields from the database is displayed to them. Some of these fields are updateable and some are not.

When the interviewer has selected the case they want to interview, they press the appropriate function key, and the case management software takes the blaise data for the case out of the BLOB field in the database and unzips it. It then creates a file called the "caseid".in file that contains case management fields that are to be updated in the Blaise data. The "caseid" is the primary key (e.g. 00000101.in). In addition to the "caseid".in file, a transaction file that instructs a manipulus shell (that is distributed with the instrument) what function to perform. The transaction file contains the primary key value for the case to be interviewed, the transaction code for the function that is to be performed, the name of the manipula script used to update the blaise data from the "caseid".in file, the name of the manipula script that will produce ASCII output ("caseid".out) from the Blaise data to the case management data at the conclusion of the interview. The layout of the "caseid".in file conforms to an ASCII datamodel that was prepared with the manipula program that updates the blaise data. Likewise, the layout of the "caseid".out file conforms to an ASCII datamodel that was prepared with the manipula program that updates the case management data.

The transaction code is used to determine which Blaise procedures to call in the manipulus shell. One transaction code is used to run the interview as outlined above, i.e. the sequence of updating blaise database - running interview – creating ASCII file to update case management. After case management processes the ASCII update file, it runs an assessor program against the control fields for the case to determine the next action to be taken. This action code is then passed back in a transaction file to update the blaise database through the same manipulus shell, but this time with a different transaction code. We also use the shell to indicate that a late mail return has been received for a case (for surveys where the CAPI work is following up on mail non-respondents), and also to indicate if a particular interview was observed by a supervisor so that the case would be screened out for potential re-interview. An illustration of this process is shown in figure 2 below:

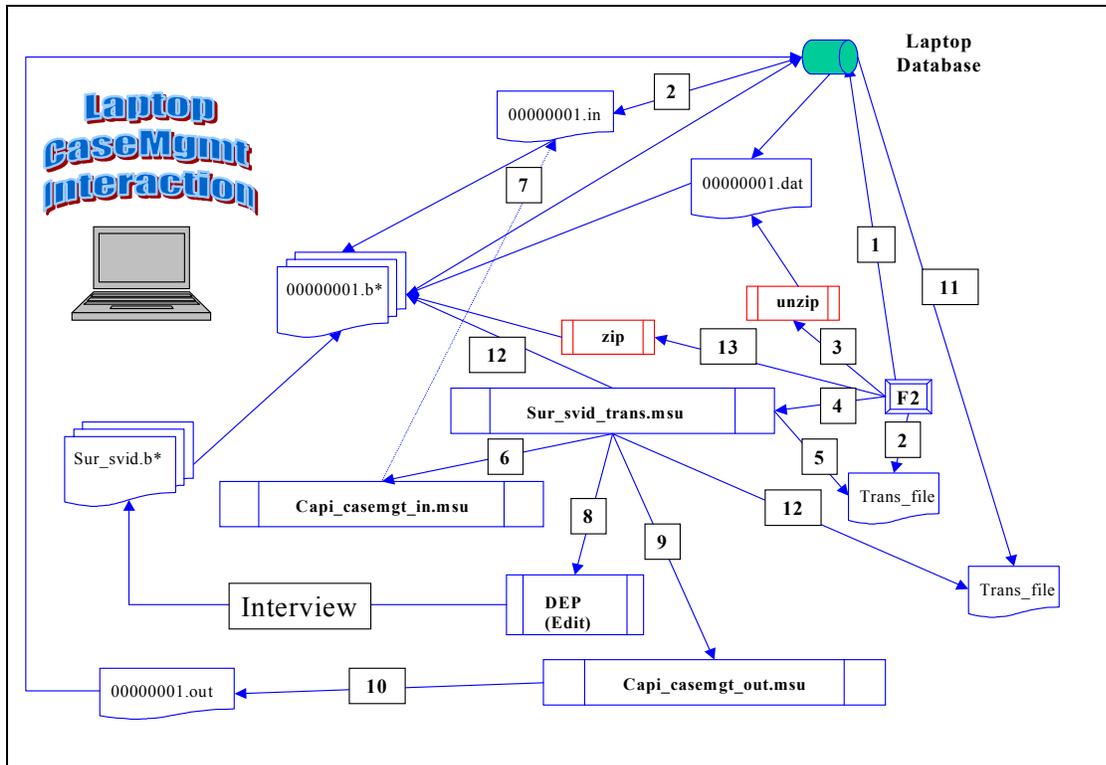


Figure 2: Laptop transaction processing

Steps in the process

1. The case management system writes the case's data from the BLOB field (zip file)
2. The case management system writes out the fields from the database that are to update the Blaise database for the case to a file named "caseID.in". It also writes out a transaction file (trans_file) that identifies the instrument meta file, the primary key for the case, the name of the Manipula script used to update the blaise data from case management, the name of the manipula script to update case management from the Blaise data, the transaction type (e.g. conduct an interview, update blaise data control fields...)
3. The case's zip file is unzipped into the appropriate Blaise files
4. The survey transaction manipulus shell is executed
5. The transaction script reads the transaction file to determine what function to perform.
6. For the interview transaction type, the following occurs: It executes the script to update the Blaise database.
7. The update script reads updates the Blaise database for the case by using the data in the caseID.in file.

8. The edit function in Manipula is then called to conduct the interview
9. After the interview completes, the script that writes out fields from the Blaise database to update the case management database is executed
10. This process produces an ASCII file named "caseID.out" which is used to update the case management system.
11. The case management determines what the next status of the case is and write out a new trans_file containing this new status (agendum)
12. The transaction script is again called, this time to update the agendum field in the Blaise data
13. Once all processes have finished for this attempt on the case, the blaise files are zipped back up and placed in the Blob field.

Some surveys allow the creation of extra units that are discovered at a sample address in the field before entering the instrument. The case management software knows which record types a particular survey uses and is able to construct an input file in the format of the original SCIF by pulling the appropriate data from the parent case. It then uses the same manipula script, which was used by MCS to load the original data, to load the Blaise database for this extra unit. It runs an ASCII to Blaise process.

Other surveys have coverage questions within the instrument that check for the existence of additional sample units sometimes referred to as "spin-offs or spawns". For example, our Survey of Income Program Participation (SIPP) interviews all the persons living in the household during the first contact. If on a subsequent contact, it is discovered that a persons living in the household at the time of the first contact has moved, the FR must attempt to locate the person that left the household. During the interview, the information for the whereabouts of the mover is obtained and those remaining in the household are interviewed as before. Once the interview has finished and the case put down, the same manipula script that writes out the control data for case management, will also create a new case for the mover. The new unit is created from the parent case with a Blaise to Blaise process.

Daily, interviewers will connect to the telecommunication server and the cases with a complete status and no further required on the laptop, will be pulled from the database and sent back. The ROSCO system will pickup the cases from the telecommunication server and place them in BLOB fields in the MCS database and update some of the control fields there as well as fields in the ROSCO database. The Regional Offices use the updated control fields in ROSCO to know the progress of their FR's work. MCS will then verify that the control data in the blaise database are consistent with the same fields in the MCS database (to verify that the BLOB field was updated correctly). For those cases where the 2 sets of control data are consistent, the MCS will produce output data for the cases that were delivered that day. It will produce either ASCII or ASCII Relational data for the completed cases, or if the data processors prefer, will create a consolidated daily file by concatenating the blaise databases for the completed cases into one. These MCS tasks are accomplished by manipula and or manipulus scripts that are run on the NT server by a remote shell (RSH) from the MCS UNIX box.

We are planning a similar approach for our CATI system except that the CATI clients in the telephone center will pull the data directly from the BLOB fields on the MCS system instead of having a database that resides locally. There will also be a CATI control table on the MCS that will contain the values of all of the control fields needed by CATI to determine the next disposition for a case. The first time a CATI workstation accesses a survey, the survey instrument will be pulled from the MCS database and unzipped on the CATI client. The instrument will remain there until the survey closes out. When the CATI interviewer requests either the next available case or a specific case, that case will be pulled from the MCS database and unzipped on the CATI client. It will then call a transaction processing manipulus script similar to the one used on the laptop, as described above. This script will read an ASCII file containing control system variables for the case that is written out by the CATI system. The script updates the corresponding fields in the blaise database files for the case and then opens the interview with the edit

function. At the conclusion of the interview, the script writes out an ASCII file of updated control information from the interview that is used to update CATI's master control file.

Manipula script generation for building interfaces

We have created generic manipulus scripts that through passing to them a set of parameters will generate manipula scripts for data interchange purposes. Some of the tasks these script perform are the following:

1. Generate ASCII output for specific fields (whose names were supplied in a file) for a specific set of cases (caseIDs also supplied in a file).
2. Update Blaise database fields from an ASCII file
3. Consolidate 2 or more blaise data base files into a single database file
4. Produce single case blaise database files from one consolidated file for all the cases contained in the consolidated file
5. Produce single case blaise database files for specific caseIDs (supplied in a file) contained in a consolidated file

Below is an example of the process for creating ASCII output for a list of control system variables:

A file containing the list of the fields to be included in the output is fed to the scripts as one of the parameters. A cameleon script, which is called from the manipulus script, will search the instrument's Meta files and attempt to locate in the blaise data model each field name passed to it. It then determines the attributes of the field and writes it to a .bla file. This .bla file will later be compiled to create an ASCII datamodel used for passing data from the instrument to the control system. In addition to constructing and preparing the ASCII datamodel for the output, for efficiency, this process also creates a filter file so that only those fields of the Blaise datamodel that are affected by the process need to be loaded into memory. So each time a Blaise instrument is opened, standard control data from the instrument is written to an ASCII file at the conclusion of the interview.

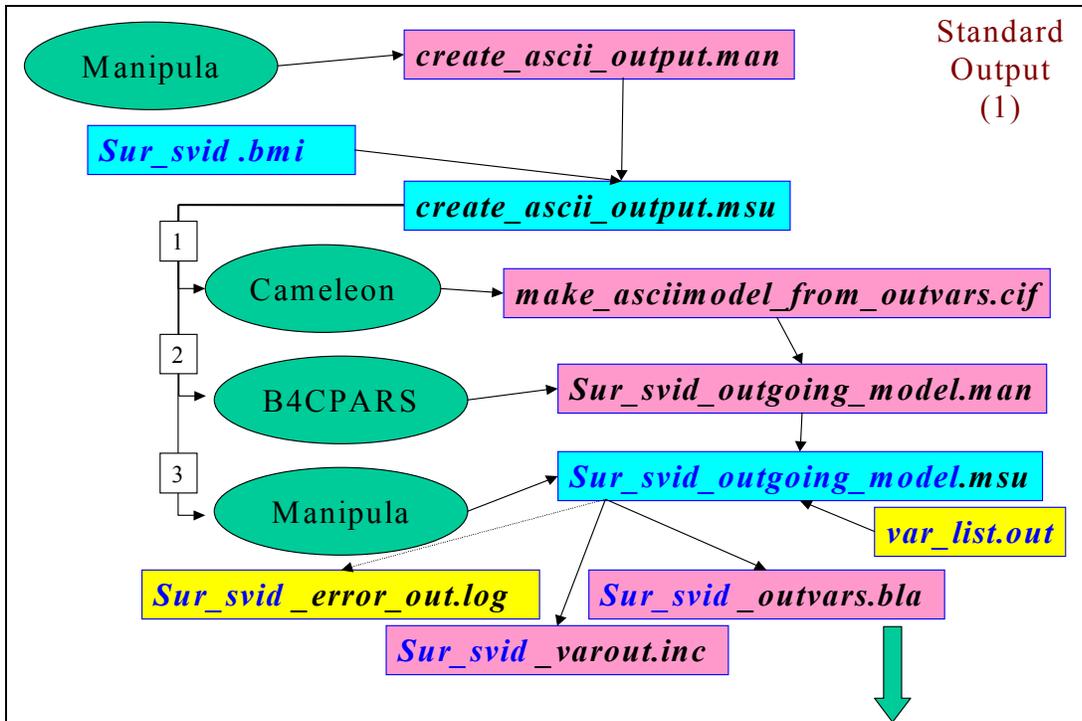


Figure 3: Creating output manipula script (part 1)

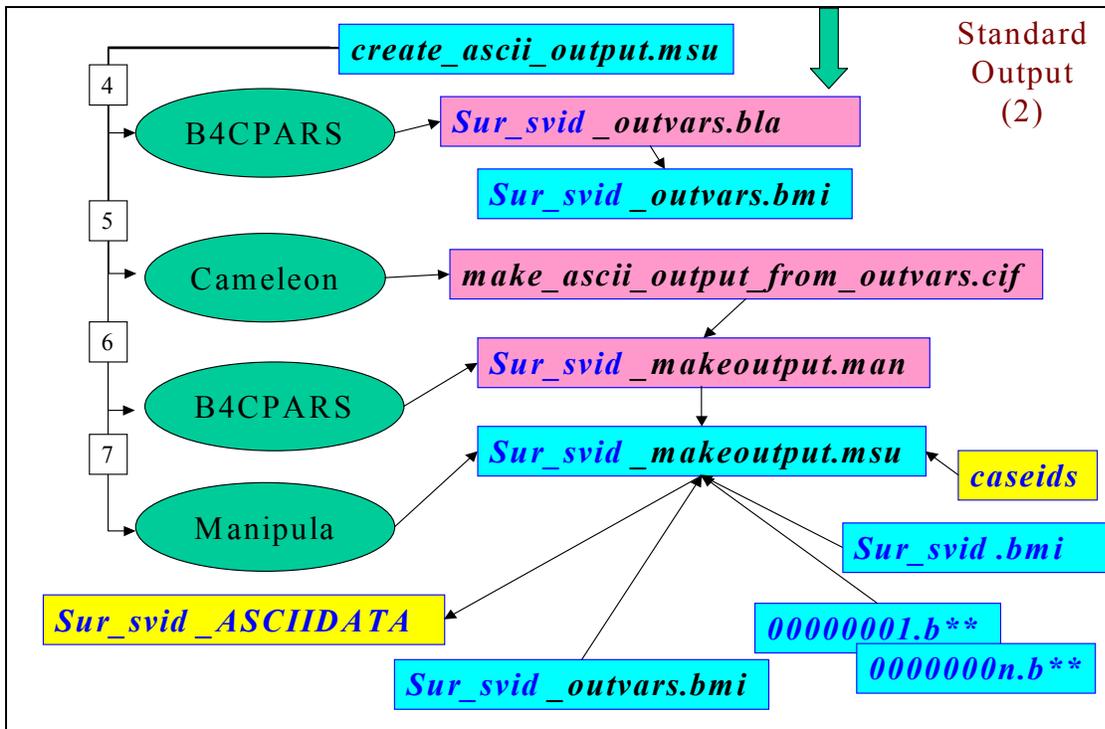


Figure 4: Creating output manipula script (part 2)

Steps in the process

1. The Maniplus shell that calls all of the subsequent procedures is written, prepared and executed
2. The Cameleon script that reads the meta definition for all the case level variables is created and executed to produce a manipula script that will ultimately compare a set of field names being passed to it against the meta data for the instrument.
3. The Manipula script created in step 2 gets prepared and executed to create a .bla file that defines the ASCII datamodel for the output file. It also creates an include file for that same set of variables, which when set as a filter expedites processing. It also writes to an error log file any field names included in the variable list file that are not found in the Blaise datamodel. In production, if there are any variables being asked for that don't exist in the blaise datamodel, processing stops until the process is rectified.
4. The .bla file created in step 3 is prepared and will be used in the last step to create the ASCII output file
5. The Cameleon script that creates the final manipula output script gets executed
6. The final Manipula script for producing output gets prepared
7. The last step produces the ASCII output by using the files created in the earlier steps. One of the parameters passed to the initial Maniplus shell determines whether or not this step gets executed.

The parameters passed to the maniplus shell are as follows:

- Parameter (1) = .bmi file name
- Parameter (2) = file of caseids
- Parameter (3) = file containing variables to be output
- Parameter (4) = location of blaise executables (drive:directory)
- Parameter (5) = location of cameleon scripts (drive:directory)
- Parameter (6) = working directory (drive:directory)
- Parameter (7) = output filename
- Parameter (8) = [X for no output] (whether or not to stop short of producing output)

Once the final manipula script has been created, it can be used to produce output on subsequent runs as indicated below.

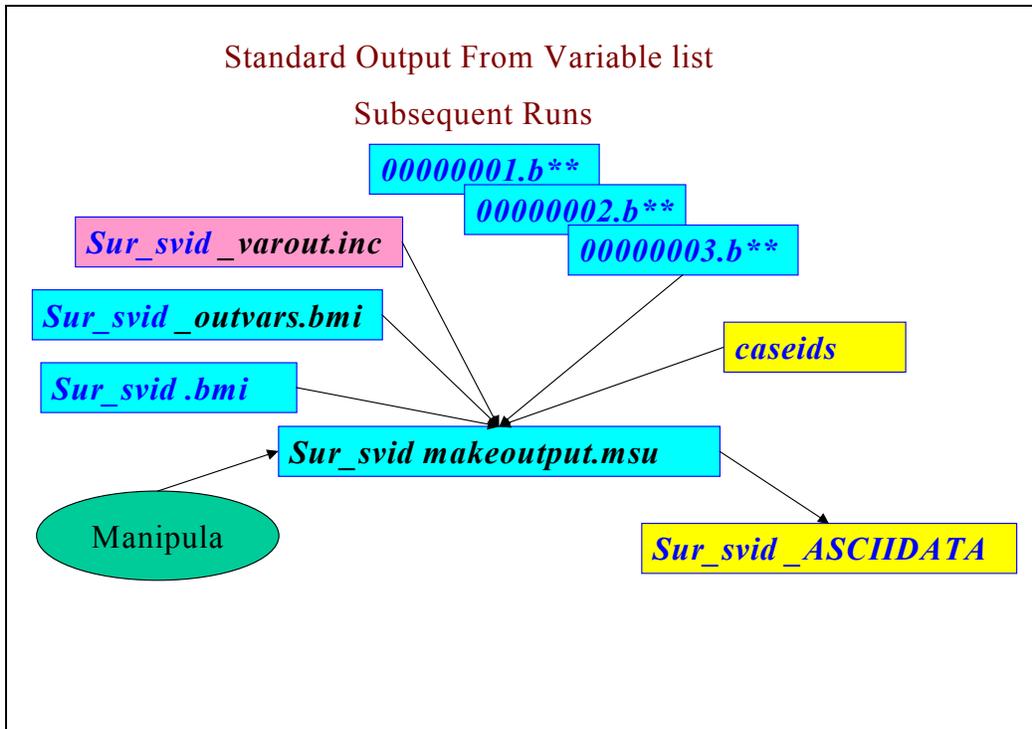


Figure 5: Subsequent Output runs

In summary, our goal is to have a standardized methodology in place to be able to field surveys in either CASES or Blaise as requested by the sponsor and to be in a position to accommodate other interviewing software as painlessly as possible should the need arise. If successful in our conversion, only those systems that will need to execute the data collection software will need to know which one to use. The Laptop case management, the CATI case management system, the Master Control and only those sponsors intending to use the proprietary data in their post-data collection processing, will need to know which software the instrument was authored in. The software for the SCS, ROSCO and those sponsor systems that are receiving only ASCII outputs should be unaffected by the choice of data collection software.

Okuka Lussamaki, Operations Research and Development Division (ORDD),
Statistics Canada, June 2001.

Introduction

Statistics Canada's Annual Survey of Manufactures (ASM) collects financial and commodities data from 35,000 Canadian manufacturers. ASM survey questionnaires are sent and returned by mail. There are four types of questionnaires: the long form, the short form, a special form for Quebec, and the head office form.

The ASM's most significant feature is the use of personalized questionnaires. The long questionnaires are personalized for each industry group and business. Personalization is based on the North American Industry Code Standard (NAICS), which determines the industry to which the business belongs. There are 23 personalized questionnaires based on the NAICS.

Before personalization, several long questionnaires are sent to businesses in a given industry. Each questionnaire contains a long list of commodities used by the industry. However, only a small subset will be used for each business based on its needs.

ASM personalization began in 1993 in order to reduce the response burden for respondents. This involved using a questionnaire personalized for each individual business in the commodities input and output sections. The long questionnaire has two parts: the financial section and the commodities section. Only the commodities section is personalized. A commodity is in fact a line with several columns where various information is collected. Historical data from the previous year are used to personalize the section. There are also extra lines where respondents can add other commodities, where applicable. About 18,000 personalized ASM questionnaires are sent.

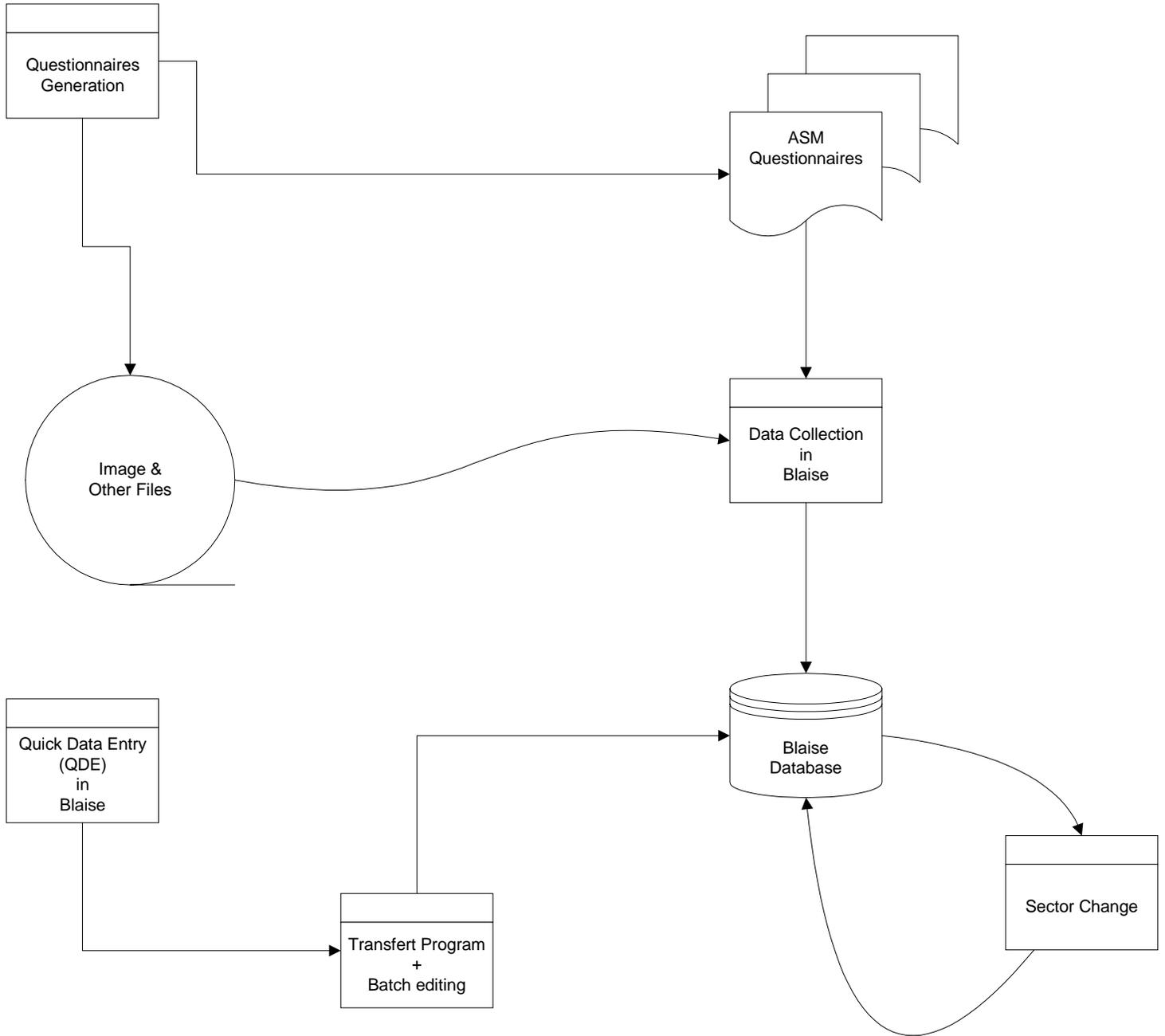
ASM Functionalities

The various ASM functionalities are:

- CATI applications: - financial section
- commodities section (personalization)
- Quick Data Entry: Blaise application for rapid data entry.
- Automated transfer: Manipula script for data transfer from Quick Data Entry to CATI applications.
- Batch editing: edits are performed in batches at the time of the automated transfer. This gives the respondent a status for follow-up at a later time.
- Change of industry: this occurs when a respondent changes industries from one year to the next.

We will review each functionality as we describe the interactions between the various applications used by the ASM.

ASM Data Collection



CATI Applications

CATI applications have two major sections: the financial section and the commodities section. The financial section uses the regular Blaise cells.

Here is an example of a financial section:

**Reporting Period Information			
<u>Reporting Period</u>			
<input type="radio"/> 1. Yes <input type="radio"/> 3. No			
QCode	<input type="text" value="0"/>		
4. From	C0011 <input type="text"/>	4. To	C0012 <input type="text"/>
5. Do the dates reported above represent a change in your fiscal year ?			
	C0059 <input type="checkbox"/>		
6. Were any of the operating units of this business unit temporarily or seasonally inactive during the reporting period ?			
	C0061 <input type="checkbox"/>		
7. Has this business unit acquired any operating units during the reporting period ?			
	C0064 <input type="checkbox"/>		
8. Has this business unit disposed of/sold any operating units during the reporting period ?			
	C0066 <input type="checkbox"/>		

Only the commodities section is personalized in CATI applications. Array structures are used to personalize the sections. As mentioned earlier, data from the previous year determine the number of commodities on the respondent's questionnaire and in the Blaise application. Arrays are filled in based on the number of commodities. Each element of the array is a commodity line, and each commodity is a block of several fields. This makes the CATI applications larger and they consequently require considerable resources, particularly in terms of memory.

Here is an example of a commodities section:

Blaise Data Entry - \NORDD10\Blaise\V44Pipes2000\WORKING\ASM2000\Leather\Leather

Forms Answer Navigate Options Help

ASM_Leather | Frame_Maintenance_802 | Appointment | Conclude | Ring_no_answer | Busy | Answering_Machine | Notes | Comments | Front_Screen | Refusal_Conversion | Key_Map

Present Contact: LUCE CARRIGAN
 Coverage: VEUILLEZ INCLURE LES DONNÉES POUR TOUS LES EMPLACEMENTS INSCRITS DANS LA SECTION F. "EFFECTIF SELON L'EMPLACEMENT".

B - Revenue
B-1 Manufacturing Outputs

3.23 Footwear (protective metal toe cap), waterproof rubber or plastics (uppers neither fixed to the sole nor assembled by stitching, riveting, nailing, screwing, plugging or similar processes)

	Commodity code for Statistics Canada use	Unit of measure	Historical Quantity	Quantity	Historical Cost	Cost at this establishment (\$'000 CDN)
3. Shipments of goods of own manufacture						
3.23	Footwear (protective metal toe cap), wat	640110000				
3.24	Footwear (without a protective metal toe	640192000				
3.25	Footwear, safety (incorporating only a p	640340110				
3.26	Footwear, safety (incorporating a protec	640340120				
3.27	Footwear, safety (incorporating a protec	640340140			20,982	19
3.28	Footwear, cowboy boots, outer soles and	640351210			1,858	1
3.29	Footwear (excluding work or cowboy boots	640351910			4,922	6
3.30	Footwear (excluding work or cowboy boots	640351920				
3.31	Slippers, with outer soles and uppers of	640359320				
3.32	Footwear (excluding sports, safety, work	640359910				
3.33	Footwear (excluding sports, safety, work	640359920				
3.34	Footwear, work (excluding safety), with	640391100				
3.35	Footwear (excluding work or cowboy boots	640391910				
3.36	Footwear (excluding work or cowboy boots	640391920				
3.37	Footwear (excluding work or slippers), o	640399920				
3.39	Footwear with uppers of leather or compo	640510000			19,589	
3.40	Liners for boots and shoes	640699910				

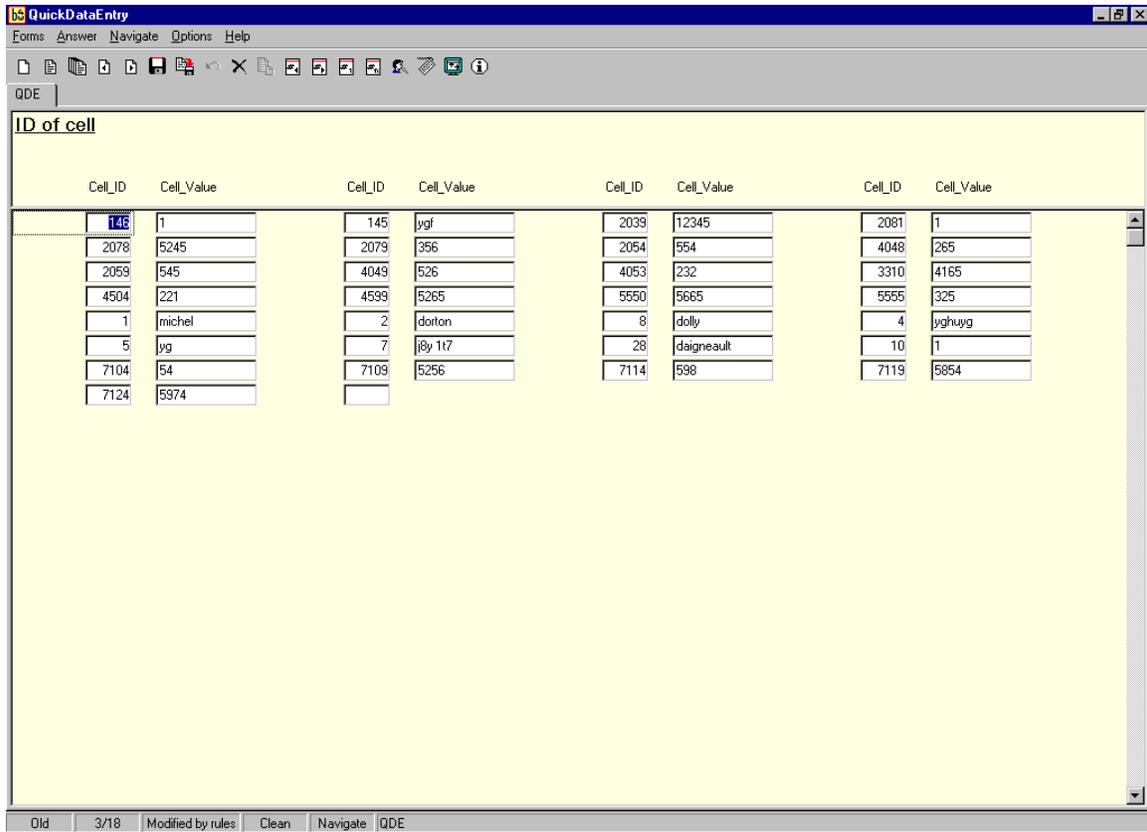
Old 17/74 Modified Dirty Navigate ASM_Leather

Quick Data Entry

Given the complexity of the array structures and their size, we thought we could create a small Blaise application (QDE) to quickly enter the data from the paper questionnaire, then transfer the data to the CATI applications.

In this application, only the corresponding cell number in the CATI application and its value are entered. This is an excellent tool for data capture. It accelerates the work of those performing data entry on the computer.

Here is an example of the data entry screen:



The screenshot shows the QuickDataEntry application window. The title bar reads "QuickDataEntry" and the menu bar includes "Forms", "Answer", "Navigate", "Options", and "Help". The main area is titled "ID of cell" and contains a table with the following data:

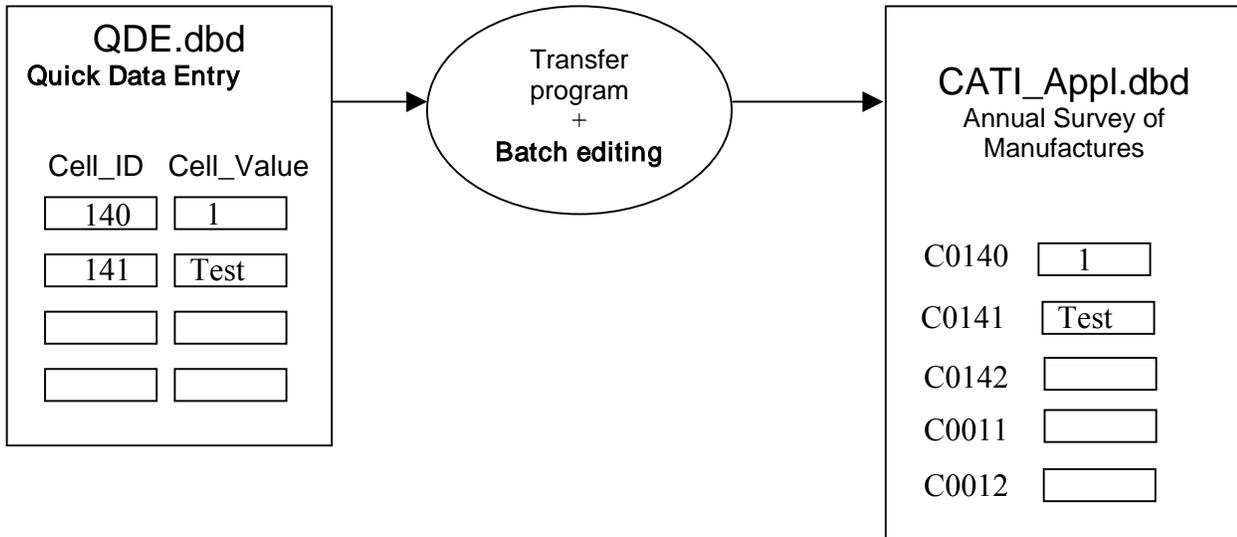
Cell_ID	Cell_Value	Cell_ID	Cell_Value	Cell_ID	Cell_Value	Cell_ID	Cell_Value
146	1	145	ygf	2039	12345	2081	1
2078	5245	2079	356	2054	554	4048	265
2059	545	4049	526	4053	232	3310	4165
4504	221	4599	5265	5550	5665	5555	325
1	michel	2	dorton	8	dolly	4	yghuug
5	yg	7	j8y 117	28	daigneault	10	1
7104	54	7109	5256	7114	598	7119	5854
7124	5974						

The status bar at the bottom shows "Old", "3/18", "Modified by rules", "Clean", "Navigate", and "QDE".

Automated Transfer

To transfer data from the heads down to a CATI application, a Manipula program is executed through a menu. This transfer allows for a search for each respondent's corresponding cell numbers in the CATI application in order to apply the captured values.

(Show an example of the transfer.)



Batch Editing

Batch edits are executed at the time of the automated transfer to give each respondent a status. The status may be final, partial, etc., depending on the manner in which edits failed. Batch editing reduces follow-up work and prevents other data entry in CATI applications.

Two essential functions are used for batch editing: CHECKRULES and DYNAMICROUTING.

The CHECKRULES function is used in a Manipula program to edit a DataModel, i.e., to scan the fields and assign their respective values without having to physically scan each form for a given survey on-screen.

When the Manipula program is executing, a second function called DYNAMICROUTING declares fields EMPTY if they are "off-route" during execution of the CHECKRULES function.

Example of DYNAMICROUTING and CHECKRULES

DYNAMICROUTING:

```
UPDATEFILE
  UpdateFile2: OutputMeta('Test', BLAISE)
SETTINGS
  ACCESS = EXCLUSIVE
  ONLOCK = WAIT
  KEY=PRIMARY
  CONNECT = NO
  AUTOCOPY=NO
  DYNAMICROUTING=YES
```

CHECKRULES:

```
MANIPULATE
  UpdateFile1.READNEXT
  WHILE NOT(UpdateFile1.EOF) DO

    IF UpdateFile2.SEARCH(UpdateFile1.QID) THEN
      UpdateFile2.READ
      UpdateFile2.Flag := 1
      UpdateFile2.Qtype:= '0014'
      PROCESS_DATA_FOOD -----> Cxxxx:=
      Cell_Value ...
      UpdateFile2.CHECKRULES
      UpdateFile2.Flag := 0
      UpdateFile2.WRITE
      {UpdateFile1.DELETE}
      UpdateFile2.RESET
    ELSE
      DAYFILE('Errors detected: ' + UpdateFile1.QID + ', ' + ', does not exist
in the database')
    ENDIF
  UpdateFile1.READNEXT
ENDWHILE
```

Change of Industry

Another Manipula program is used when a respondent changes its type of activity (for example, when a fabric manufacturer becomes a clothing manufacturer).

This business changes its code under the NAICS and thus requires reclassification in another industry. The change of industry must be indicated in a CATI application in order to change the NAICS and the transfer flag. The business must therefore be transferred from the database of its former industry to that of its new industry under the NAICS.

The Manipula script executes this transfer while respecting all the new changes. It uses a flag to search for the cases to be transferred, then finds the databases for the new NAICS.

Since it is a business in a new industry, the respondent is sent a new questionnaire for that industry. The questionnaire contains the first 10 commodities for each section associated with this industry. This will allow for personalization of the questionnaire and will reduce the response burden for the respondent. Extra lines without commodities are available for the business to make any necessary additions.



Problems

The following problems were encountered during application development:

1) Screen Personalization

Since we are using array structures for personalization, we had to display only those elements of the array that were filled in. Three problems arose.

- First, Blaise does not allow for the use of dynamic array structures.
- Second, to manipulate the arrays, we had only the loop "FOR" for personalization. Because the array is static, we could not scan it entirely for the on-screen display. Moreover, the loop "FOR" does not allow for an "Exit" when a condition is met. This is possible with Manipula but not with Blaise as yet.
- Third, the commodity descriptions and codes had to be read from an external database. We could have automatically stored them in the applications when the historical data were loaded. But the problem was to display all the fields the client needed to see on-screen. The major difficulty was that these had to be displayed in English or in French, according to the language used by the respondent.

To solve these problems, we began by using one field for each personalized section that would contain the number of commodities recorded the previous year. This field was assigned a value when the historical data were loaded in the database. Thus, using the loop "FOR", the limit to be verified was the number of commodities, which made it

unnecessary to scan the entire array. At the same time, this allowed us to display only the required number of commodities. Simulating a dynamic array thus solved the first two problems.

For the third problem, we also created a flag to find the commodity description and code in the external database while we scanned the table of commodities. We also put the external database in read-only mode to accelerate our search.

2) Edits for Commodities

The array structure meant that no cell numbers were associated with the commodities. This posed a problem for identifying our edits, i.e., which had failed and which had not. Since these rules were almost the same for all commodities, it occurred to us that we could create generic rules. To identify these from one commodity to the next, we added a field that contains the number associated with each cell entered for commodities. This number is generated only if the cell is entered.

For example, for commodity 10, we entered “20” for the quantity and “1000” for the cost.

		Commodity code for Statistics Canada use	Unit of measure	Historical Quantity	Quantity	Historical Cost	Cost at this establishment (\$'000 CDN)
3. Shipments of goods of own manufacture							
3.27	Logs of spruce, pine or fir (SPF) combin	440320991	Cubic metres.		20		1,000
3.35	Logs of poplar, aspen or cottonwood	440399800	Cubic metres.				

The associated numbers will be, for example, “23010” for the quantity and “25030” for the cost. The edits will thus take these numbers into account in order to distinguish them from one commodity to the next.

3) The Length of the Arrays

The length of the arrays presented a problem since it slowed down the applications. Their length varied between 100 and 350 elements from one application to the next. The elements of these arrays are blocks containing several fields.

For example: `ArrayCommodity : ARRAY[1..350] OF Bcommodity`

It took from one to two minutes to go from one cell to the next. When we asked our clients for the average number of commodities filled in for all applications, the answer was less than 100. We shortened our arrays, which dramatically improved the performance of our applications. We also used a special layout to display less than 20 commodity lines. Changeover time from one cell to the next varied between 4 and 6 seconds.

One fact that should be noted here is that static arrays are a substantial problem with respect to unused memory space. This is particularly so in the case of very large arrays, as with Statistics Canada’s ASM.

Conclusion

All ASM functionalities are managed through a menu. Other changes can be made if Blaise is improved, particularly with respect to arrays, which are currently static and take up a lot of memory space. If dynamic arrays were used, CATI applications would be more powerful and would require less memory. This would also reduce access time to very large databases since we now experience delays when our databases grow larger. Another solution would be relational databases, which would allow for better management of large Blaise databases. We hope to be able to achieve all this with new versions of Blaise.

Case Management for Blaise using Lotus Notes

Fred Wensing, Australian Bureau of Statistics
Brett Martin, Statistics New Zealand

Introduction

A significant aspect of field based interviewing is the need to manage the distribution of cases to the interviewer workforce and then to provide those interviewers with a convenient interface where cases can be viewed and interviews initiated.

Although the Blaise suite provides a range of software which can be used to build a suitable Case Management system, there was a preference at the Australian Bureau of Statistics (ABS) and Statistics New Zealand (SNZ) to develop such systems using facilities which are more closely aligned with other systems used in the office. The arrival of the Blaise component pack, and the Blaise API in particular, has made it possible to develop such systems in software other than Blaise while still being able to communicate directly with Blaise data files.

This paper describes the Case Management systems being implemented by ABS and SNZ using Lotus Notes, as the main office based software, and combining this with Blaise to achieve fully integrated facilities to support survey operations in the field and also in the office.

Features of Lotus Notes

Lotus Notes provides Email, workgroup collaboration, document management and Internet services using an intuitive browser-like interface. Notes can deliver significant functionality, security and customisation with applications that range from standard messaging, through collaborative discussion and document libraries, to tailored applications. Lotus Notes is available to all ABS and SNZ staff and is used extensively throughout both agencies.

Notes databases generally contain information about a single area of interest. While the documents within a Notes database can be considered as "records", such documents are more sophisticated than typical database records in that they can contain rich text, pictures, objects and many other types of information.

In recent years, in both the ABS and SNZ, Lotus Notes has emerged as the main application software for many administrative and statistical processes that are carried out on a day-to-day basis. One reason for this is that applications written in Lotus Notes have the same look and feel as the Email and document systems which staff use every day. Notes based applications can therefore be more readily integrated into the desktop environment.

Two significant features of Notes make it an attractive option for Case Management. These are its embedded security options and the ability to efficiently replicate information across the office network, or through the Internet to mobile computers in the field.

The security features of Notes involve a level of encryption which ensures that data is protected from unauthorised external access while on the computer or during transmission. Notes also provides a comprehensive system of access controls which can be used to restrict access to processes, views, forms and data fields.

The replication features of Notes ensure that data which is exchanged via the replication process is successfully updated without risk of corruption. If the same data has been updated at both ends of the replication then both copies are exchanged and identified as a potential conflict. Replication can also occur selectively so that only data which meets selection criteria and security criteria is exchanged. The Notes replication process is efficient in that it only exchanges updated data at the field level within a document, thereby avoiding the unnecessary exchange of unchanged information.

Case Management configuration under Lotus Notes

The main features of the Case Management systems being built in Lotus Notes at ABS and SNZ are:

- a simple graphic user interface for interviewers;
- a Notes database which provides a space in which the cases are presented, distributed and managed;
- Notes documents which serve as containers for case identification, and subsequently for the completed Blaise instrument databases;
- case workflow is managed through the status of the Notes document in the database;
- case information (including partial or fully completed Blaise instruments attached to the Notes document) is exchanged between the office and the field through a system of selective replication;
- various processes are activated via buttons and menu options which execute particular scripts;
- key fields and status information are exchanged between Notes and Blaise through the API;
- security and access control settings ensure that operators can only view and work with cases that are assigned to them;
- use of summary views of the Notes documents to provide up-to-date status information.

These are described in detail below.

Simple Graphic user interface for interviewers

Notes provides facilities to develop graphic user interfaces which contain hotspots, or shortcuts, to Notes databases and other processes. This makes it possible to provide staff with a simple Web-like interface that gives direct access to the main survey processes. Such interfaces are particularly useful for field staff who may not be accustomed to the busy screens often found in today's desktop computing.

Figure 1 shows the graphic interface or "Home page" developed for field staff using the SNZ Case Management system. A similar interface has been built for the ABS system. The page displays only a few choices, which are easy to recognise, and each choice contains a link or shortcut to the relevant process.

Figure 1. Home page for field staff



Notes database

The Case Management system resides within a Notes database which contains forms, views and scripted processes to facilitate all the operations required.

Access to the database is controlled through standard Notes access controls in which users are defined by:

- user type - person, group, server
- access level - reader, author, editor, manager
- role - office user, administrator, interviewer, supervisor

All users require a current Notes user certificate issued by the agency before the Notes system can be activated. All users are required to supply a current password before being able to proceed. The default access to the database is "No access" which prevents unauthorised officers within the organisation from accessing the information. The Notes database can also be stored in an encrypted form so that data is protected from unauthorised access from outside of the Notes system.

Apart from these general access controls, further controls on access are contained within the various components of the database to restrict various functions to relevant users (see later paragraphs).

The database contains a general profile document which is used to record certain system metadata such as the path names for the system software (like Blaise or Winzip) for both office and field environments, as well as the file types which make up a Blaise instrument. The profile document is activated whenever the database is opened and enables the processes to be responsive to changes that may be required from time to time.

A tailored document is used to record the metadata (name, identifier etc) for each survey to be operated through the database. This document also stores the Blaise metadata files required for the interviewing process as attachments. These are detached by the system to a local disk drive for use when required.

All documents are presented through a system of views which show a selection of documents that are sorted and grouped according to relevant characteristics. Figure 2 shows a typical view of cases in the SNZ Case Management system. Similar views have been used in the ABS system. Each row in the view represents a single case and a selection of the characteristics, some in the form of graphics, are presented to assist the user in selecting the one to act upon next.

Figure 2. View of an interviewer's workload

Scheduled	Time	Survey	Reference Number	#	Status	Date Due
Today	13:45	HSS	R95308002	1	Active - Not Started	31/07/2001
Today		HSS	R95308005	1	Active - Incomplete Non-contact: try again	31/07/2001
Today		HSS	R95308006	2	Active - Incomplete	31/07/2001
Today		HSS	R95308008	2	Active - Incomplete	31/07/2001
Today		LISNZ	R75293003	1	Active - Not Started	31/07/2001
Today		LISNZ	R75293004	1	Active - Not Started	31/07/2001
Tomorrow	11:00	HSS	R31997001	1	Active - Not Started	29/09/2001
17/09/2001	18:00	HSS	R31997003	1	Active - Not Started	29/09/2001
		HSS	R31997002	1	Done - Complete	29/09/2001
		HSS	R95308001	1	Received - Complete	31/07/2001
		HSS	R95308007	1	Received - Complete	31/07/2001
		LISNZ	R75293001	1	Received - Complete	31/07/2001
		LISNZ	R75293002	1	Done - Complete	31/07/2001

Notes document as a "container" for case information

The main container for case information is a Notes document which has been designed to be as generic as possible so that it can serve all types of surveys.

The document has been set up to contain the following:

- case identification;
- street address information;
- the name of the instrument to be used for the interview;
- the name of the interviewer to whom the case is assigned;
- status information to indicate where the case is and whether interviewing is complete;
- fields for the details of the survey respondents;

- fields to be used in any office processes such as coding of occupation or industry;
- buttons that can be used to initiate various processes for both office and field work.

Figure 3 shows a typical document containing case information from the SNZ Case Management system. Similar documents are used in the ABS system. The document shows the essential fields necessary for the field staff to use in their work. Other fields, such as the instrument name, are hidden from the interviewer but are accessible to the system. Buttons or hot spots are used to activate particular processes.

Figure 3. Document containing case information

Case Information - Lotus Notes

Close

HSS R95306008 - 8 lbug Way **Case Active**

Today 20:00 **Period Start** 01/07/2001 **Due** 31/07/2001

Household	Complete: Response
Fred	Not Started

Visits

17/07/01 13:00

31/07/01 13:00

Visit time: 00:01:48

Edit time:

Brett Martin's Notes

Respondent does not want to start interview until this evening

Region 7 Supervisor's Comments

The case documents are created from lists of selections provided from the sample frame (held in agency databases) through office processes that are activated from menus or action buttons in the database. The system also allows for a set of generic training cases to be loaded into the database and assigned to interviewers for training purposes.

When an interview is started for the first time from within a case document, the system creates a Blaise data file to record the interview which it places onto a working directory on the hard disk. It does this using the instrument named in the case document along with the metadata files located in the survey definition document (also detached).

Various fields in the document, such as the respondent details and coding fields, are populated with data from the instrument (through the Blaise API) whenever interviewing is carried out or completed. Similarly, aspects of the case which are changed in the Notes document are transferred to the instrument (through the Blaise API) the next time it is activated.

At the end of an interview the Blaise data files that were created or updated are closed and attached (or reattached) to the relevant case document.

Each case document contains all the information about the status and content of one case. This makes it easy to move a single case through all the stages of the survey collection process.

Workflow managed through the status of the Notes document

Once all the case information is placed into Notes documents it is possible to see the survey situation through a series of views which present lists of those documents, selected and sorted according to specific criteria. The key criteria, as mentioned above, are the status fields which reflects the current state of a case.

The status fields in the document will typically go through the following states:

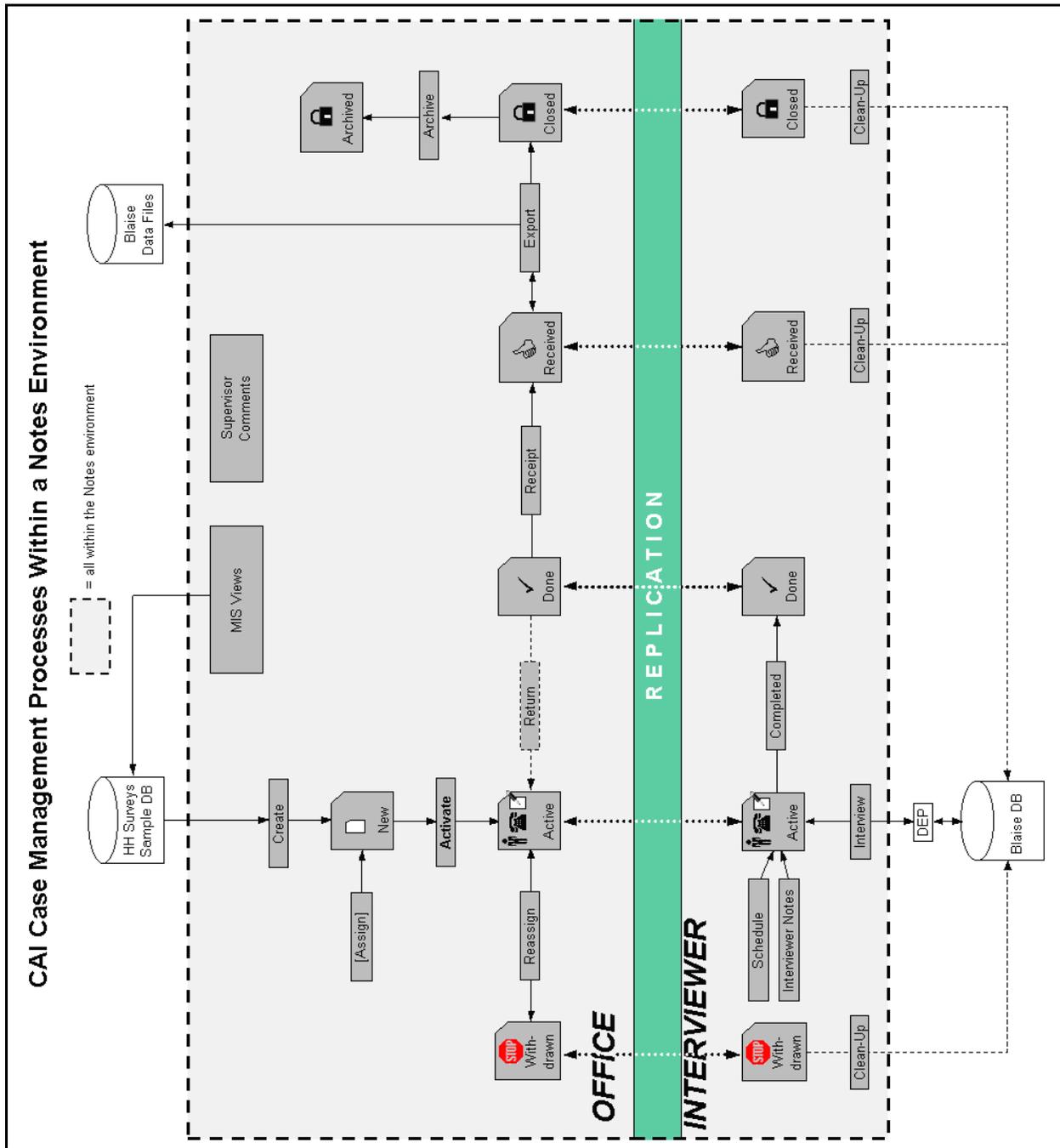
- New – document created for a selected address (ready for assignment);
- Active – available to the interviewer;
- Done – interview completed and ready for return to office;
- Received – returned and accepted by the office;
- Reassign – for assignment to another interviewer;
- Withdrawn – removed because it was reassigned;
- Closed – exported from the system to further processing;
- Archive – removed from the system after export.

Where additional processes are to be carried out, such as office coding, then an additional state can be added to cover that.

Depending on the status of a case, the system enables particular functions to occur or prevents them from occurring. In that way the cases can be assisted to "flow" through the system. For example, when the status of a document is set to "Active" it becomes available on the interviewer's computer and interviewing can be carried out. Once the status of a document is set to "Done" in the field then interviewing is no longer possible.

Figure 4 shows how document status can be used to define the workflow from creation through all stages of collection to archival. At various stages the document, containing the case, is replicated to or from the interviewer in the field.

Figure 4. Case Management workflow



Case information exchanged through a system of selective replication

As mentioned earlier, one of the main features of Notes is the ability to replicate or exchange data between copies of a database. In the case of the management of a survey in the field, every interviewer is equipped with a replica of the main Notes database but the replication settings and document security ensure that interviewers only receive the cases which are intended for them and no others.

Selective replication is important for reducing the volume of data which is exchanged and reducing the risk of an interviewer accessing the wrong records.

Selective replication works through applying a selection condition to the documents which are exchanged between the office copy of the database and field copy. The selection conditions and document security settings identify the type of documents to exchange and/or whether the documents have been assigned to the interviewer (identified by their logon user name) who is carrying out the transmission.

Processes activated through buttons and menus

The Notes database environment is one in which the user can be provided with a simple and user-friendly interface. Most processes that need to be carried out are made easy to do through a system of buttons and menus which are located in relevant positions throughout the views and forms.

The environment for field staff is made even easier by setting up the system to "hide" views and processes which are not relevant to them. Some standard features of the Notes environment can also be turned off for the field staff who do not need to use them. Other simplifications are possible through "automation" whereby processes are triggered whenever certain circumstances arise. For example, a new survey instrument is detached from the survey metadata form the first time it is needed.

The environment for office staff, while more complex than for field staff, is expected to be relatively easy to use by them because it conforms in design and features to the standards and conventions in place for the many other Notes based applications that exist at ABS and SNZ.

The buttons and menus provided in the database activate various agents which have been written in Lotus script and are stored within the database.

Execution of Blaise processes under Notes

Where processes, such as data capture or export of data, need to make use of Blaise software components these are executed from within the Lotus script. The relevant call to the software is prepared within the script and then executed directly. Control of the process is handed to the relevant Blaise executable through the use of the Windows Scripting Host Shell method. When the particular process is completed the control returns to Notes.

The availability of a Dynamic Link Library (DLL) in Blaise 4.5 makes it possible to call the Blaise Data Entry Program or Manipula in a direct way.

Key fields and status information are exchanged between Notes and Blaise through the API

Where needed, the scripted agents access the Blaise data files for any case through the Blaise API.

The Blaise data files are accessed from a working directory on the hard disk where they are placed when the case document is opened. When access to the Blaise data is completed the data files are then reattached to the case document in the Notes database. The metadata definition files are not attached to case documents since that would result in many copies of the same metadata files being stored unnecessarily.

In order to facilitate the exchange of information between Notes and Blaise a standard list of fields to be exchanged was developed in Blaise and is incorporated into every instrument. This makes it easier to use the same agents for many surveys and reduces the maintenance requirements.

Access control settings to ensure operators can only access their own work

As mentioned earlier, the general access and security of the Notes database is managed through the Notes user ID and access settings in the Notes database. Access controls go further than that, however, to ensure that operators only access the cases and functions to which they are authorised.

Access to elements of the Notes database (views, documents or functions) is mostly managed through the role setting. Office staff who need access to the full range of functions are added to the access control list by name and then placed into the relevant role. This then opens up relevant parts of the database for them to use.

Particular functions are enabled or hidden for particular roles in the database through the use of "Hide when" statements which can be attached to any element of the database (i.e. to views, forms, fields, buttons, or even data fields within any document).

Cases are assigned to interviewers by their login name. Once that assignment has been made then view settings and/or document security settings are used to enable or restrict the views of available cases.

When combined with the selective replication and security settings mentioned above, the Notes database environment provides a shared workspace in which each user gets to see and act only on those documents which are of direct relevance to them.

Summary views of the Notes documents provide up-to-date status information

With all the case documents being stored in the same Notes database it provides a convenient place to obtain up-to-date information on the progress of the survey.

Status reporting is achieved by a series of summary views through which counts and summary totals are produced for the main status fields. Totals can be presented by various items of interest such as by interviewer or region. Whenever the case documents are updated the summary views will immediately reflect the latest position.

Discussion issues

The implementation of any Case Management system has to handle specific problems. The discussion which follows looks at the way that these issues have been addressed in the context of Notes

Concurrency control of cases

In any computer based Case Management system there will be many times when there are two copies of a case. The most common situation occurs while a case is in the field and the original copy is still in the office system. If updates are made to both copies of the case, problems can arise when the two copies are brought back together. Firstly, the system needs to detect that changes have been made to both copies and, secondly, a decision needs to be made as to which one should be kept.

In relation to the first problem, Notes detects whether changes have occurred in two (or more) copies of the same document since the last time they were replicated. This is a standard feature of Notes and when potentially conflicting changes to a data field are found a duplicate document is produced and flagged by the system as a "replication conflict". Unfortunately, resolution of the conflict still requires someone to make a decision as to which copy should be kept.

The Case Management system described in this paper has been designed in such a way that conflicts of this kind are very unlikely to occur. By using the document status outlined earlier, along with Notes access controls on documents and functions, it is possible to prevent changes made in the office copy of a case document conflicting with changes made to the same case in the field.

Case reassignment

One common situation, which can lead to a conflict of cases, if not managed properly, is the reassignment (or transfer) of cases between interviewers. If cases are simply reassigned and sent to another interviewer, then a “replication conflict” is very likely to happen, particularly if the first interviewer has already commenced an interview.

The solution developed for this situation in Notes is for case documents that have been marked for reassignment to be duplicated by the system and then the new copy is assigned to another interviewer. The old copy of the case remains but has its status changed to “withdrawn” which will eventually lead to its removal from the system.

Creation of new cases

From time to time it may be necessary for new cases to be created in the field, particularly to handle variations in the sample since address lists were prepared.

The system includes agents which enable an interviewer to initiate a new case. Details of any new cases are then transferred to the office through the replication process

More than one interviewer using the same computer

Systems need to accommodate the situation where more than one interviewer can use the same computer. Certainly this is the case in the ABS where computer assisted interviewing is still being introduced and not every interviewer is equipped with a computer. While it is possible to configure a computer for two or more users, the problem is to ensure that the Case Management system presents only the relevant cases to each interviewer while using the same facilities.

As mentioned earlier, the Notes system for Case Management employs settings within the database that ensure cases are only visible where the assigned interviewer name matches the interviewer’s logon name. Thus two (or more) interviewers can safely work within the same database on the same computer but access only the cases which are assigned to them. The only problem that remains is that each interviewer will need to replicate the database with the office in person because the selective replication settings will only exchange cases that relate to whoever is logged on at the time.

More than one survey to be supported at the same time

The Case Management system described above has been implemented to be as generic as possible. The parts that change from survey to survey (eg. the instrument) are managed through the survey metadata document. Most views of the cases in the database also include the survey name as a key attribute. It is therefore possible to support the operations of more than one survey at a time.

Support for multiple surveys extends to the interviewers, who may be called upon to work on different surveys in the same time period. When that happens, the views in the database will show the cases grouped by survey name if required.

Avoiding data loss and corruption

Systems that rely on field computers are liable to place the data at risk of loss or corruption due to events such as loss of battery power or interviewers not following procedures. It is necessary to build remote systems in such a way that interruptions to the interview do not place the data at risk.

The Notes based Case Management system described above uses a local directory to place the Blaise data files that are used for interviewing. At the end of the interview the data files are attached or reattached to the case document. In order to avoid potential corruption or loss of the data the following measures were implemented in the actions associated with interviewing:

- data files are archived before and after the interview is carried out – this ensures that recovery of at least some of the data is possible;
- the working directory used to store the data files during the interview is examined for any residual files from earlier interviews before a new interview is started. If files are found then special recovery agents are activated to locate the case relating to the earlier interview and reattach the data files;
- while the interview is in progress other functions of the computer are suspended until the interview is terminated in some way – it is not possible to make changes to any other cases while an interview is in progress.

Transmission and security issues

Good case management goes beyond the management of cases in the office and the field to include the secure transmission of case material between the office and field work force.

The Notes environment uses standard Internet TCP/IP transmission protocols. Industry standard encryption is employed both within the Notes database and during transmission to ensure that data exchanges are secure. Furthermore, interviewers can initiate the transmission process through the simple press of a few buttons.

While the level of security provided in this way is sufficient for safe transmission via Notes and the Internet, an additional encryption and authentication layer can be employed through application of Virtual Private Network (VPN) technology if so desired.

All data stored on the hard disk in field computers is also held in an encrypted format available under Windows 2000.

Software upgrades

All computer based systems need to be designed for the possibility that software will be upgraded from time to time. The software component which is regularly updated, of course, is the survey instrument but major software upgrades also need to be handled.

The Notes Case Management system described above has been built to handle a range of different survey instruments, although they are required to have a common core of management fields. New instruments are defined and attached to a survey metadata document, as and when required, and replication takes care of their distribution to the field.

Support for major software releases to field computers at ABS and SNZ is also managed through a Notes database application. Given the potential complexity of this process, however, the application is kept separate from the Case Management system, although there are important links. Briefly, the software release application maintains a profile document for each field computer and updates are supplied through “container” documents similar to those used for Case Management. When an upgrade is required the system detaches the software and places install agents in the start-up folder to be activated when the computer is next switched on.

Email, news and other applications of Lotus Notes

The use of computers that are configured to run Lotus Notes opens up many other possibilities for information exchange by field staff.

While direct use of Email through Notes is easy, it can lead to problems with foreign material entering systems. As a result, there is a preference not to provide full access to Email services to field staff. Instead, Lotus Notes can provide the ability to exchange information through a “discussion” database where office and field staff can create and exchange documents in a shared space for all to see. This database is also kept separate from the Case Management application.

Notes applications have been developed at ABS and SNZ for other administrative processes associated with field staff. These include Contract management, Reference information, Time and travel recording and Pay advice.

Conclusions

Case management is a vital part of survey operations and it is important to have a system that can support the wide range of functions that case management requires.

This paper highlights the features of Lotus Notes which make it an excellent choice of software to build such a system. At the same time, there is recognition that the existence of Blaise API components has made it possible to build a flexible system which integrates the good features of Notes with the strengths of Blaise in conducting interviews and storing the data.

Acknowledgements

The authors of this paper would like to acknowledge the work of Reece Guihot and Michael Booth at ABS, and Sean Keefe and Martina Wynen at SNZ, who have contributed significantly to the development of Case Management systems in Lotus Notes at both agencies.

LWR: An integrated software system for the household expenditure survey (Continuous economic computations [German abbreviation: LWR])¹

**Thomas Pricking, Landesamt für Datenverarbeitung und Statistik
(Federal State Bureau of Data Processing and Statistics)
North Rhine-Westphalia (LDS NRW), Germany**

1. Preamble

Perusal of the documentation created following the Blaise Users Conferences quickly discloses three extremely comprehensive articles on the subject of *Household Expenditure Surveys*. The central emphasis of this observation varies in this context from the use of encoding methods (ROESSINGH et al., 1993), via comparative assessment of the PAPI and CATI surveying procedures (MANNERS, 1995), up to and including the use of Blaise III as software for the integration of a range of different surveys (MANNERS et al., 1997). SCHWAMB and WEIN (1997) also examined this subject. The question arises in view of these publications of whether there is anything new to report at all, and if so, what can be said on the subject of *Household Expenditure Surveys* which is new, from the point of view of Blaise programmers and users.

The term "new" can be viewed in this context from a number of different angles: On the one hand, there is the technical viewpoint, as to whether new potentials of the Blaise development tools are exploited, for example. I intend to take only the cursory look at this subject. On the other hand, the taking of particular account of users with their special needs and tasks in the development of a software package can result in a new mode of observation. This is precisely the focus of this essay: Is it possible using Blaise to provide specialist statistical departments with support in such a way that account can be taken of complex conditions and extremely heterogeneous requirements? The basic statistical fundamentals are firstly described, in order to render planning activities and their implementation comprehensible in the form of a **experience report**; the basic outlines of the statisticians' list of requirements is then examined, and the solution finally presented.

2. Statistical fundamentals

The statistics of ongoing household computations have been performed at regular intervals in Germany since 1950. Throughout Germany, 1000 households initially took part; these were joined after re-unification as from 1993 by a further 1000 households on a voluntary basis, categorized into three household types: Employees' households with average income levels and government officers' and salaried staff households with higher income levels, each consisting of four persons, and two-person households in the form of pensioners and recipients of social assistance. These households kept a monthly household ledger in which all incomings and expenditure were recorded in minute detail. The households generally remained in the random sample for a number of years, in order to ensure the most continuous flow of data possible. Data preparation in the statistical bureaus was accomplished using traditional means: The dossiers received were checked manually, and allocation of the products acquired by the household into the books was performed by hand. Once the central data acquisition elements had entered the comprehensive survey data into a mainframe computer, the data was ordered in batch runs and manual intermediate steps. The survey was administered by means of fling cards.

¹ This article and the corresponding presentation are dedicated to Bernward Hausmann, who continuously promoted the use of Blaise throughout his period of service with LDS NRW.

This procedure, which was an accepted practice in the 1960s and 1970s, had fallen, by the 1990s, at the latest, into technical and organizational disuse: The not inconsiderable burden placed on the households led to a drop in their willingness to participate, which for its part produced a decline in representative quality. The period between the performance of the survey and publication was considered excessively long, and the processing phase to be of poor effectiveness and efficiency.

For this reason, the German Office of Statistics decided in 1994 to overcome these difficulties by means of a new methodological concept. The following restructuring measures were to be implemented:

- New survey ledgers were intended to make it easier for the households to provide answers to the comprehensive list of questions.
- A rolling quarterly panel was created, in order to reduce the burden on the participating households. The random sample was now to consist of 6000 households, each household needed to respond only once each quarter, however. This provision was intended to retain the legally specified scope of random sampling (2000 households) and, at the same time, achieve a broader scatter of households.
- The constitution of the random sample was to be placed on a new basis, and, therefore, to achieve greater representative quality, via orientation around other socio-demographic facts derived from the micro-census.
- Technical support was to be placed the completely new foundation. The batch-based procedures were to be superseded by dialog-based solutions, with comprehensive support using computer-based means targeted for all working processes.

The feasibility of the new conception was examined in a pilot study in 1996 and 1997 (see CHLUMSKY et al., 1997 and GERTKEMPER, 1997). In view of the positive results obtained, LDS NRW was awarded in 1998 a commission by the Statistische Verbund for the drafting of the programs necessary for the new procedures.

3. Requirements on the computer system

A data-processing project is generally preceded by an assignment analysis, in order to determine the subsequent users' technical and organizational needs. Two specialists, from our own statistical bureau, on the one hand, and from the Federal Bureau of Statistics, on the other hand, were available as contacts for the four-person development team. Personal know-how acquired in the context of test surveying, was not available for use, since both bureaus had implemented a change of personnel as a result of internal restructuring provisions in the field of economic computation. In addition, it quickly became apparent that the programs produced for the test survey had largely not subsequently been used, for a range of different reasons. In short: It was necessary to develop and program a completely new data-processing system. Luckily, the concepts of the technical representatives complemented each other ideally: The Federal Bureau of Statistics placed particular emphasis on the taking of random samples, plausibility checking and drafting of results, whereas the manner of observation of the LDS representative concentrated on survey organization and on the practical usability of the programs. Nine modules which would need to be created became apparent relatively quickly:

- a) Address management for interested and participating households;
- b) Random-sampling takings;
- c) Survey management;
- d) Acquisition and plausibility checking of the statistical data;
- e) Setting-up and upkeep of a classification system (COICOP);
- f) Budgeting of the households' income and expenditure;
- g) Classification of the households;
- h) Exportation of the ordered data for determination of weighting factors;
- i) Re-importation of the weighting factors and exportation for tabularization.

The LDS NRW had defined for the test survey a base module **address management**. It was, it is true, possible to integrate this module into the basic structure, but it needed to be augmented with significant elements. It was necessary to integrate both an importation program and a checking for duplication, since use was to be made for the survey of existing addresses of the households which had already made up the reporting group or had participated in random sampling of income and consumption and, in addition, further households were to be recruited by means of advertising provisions. In order to simplify searching in the address basis, further, secondary keys were generated in addition to the household number constituting the primary key. The post-code directory and bank sorting code directory were, of course, integrated as coding files, in addition to a range of export formats, in order to facilitate further utilization of the address data in the form of multiple letters, label stickers, etc..

Although it was possible to implement these provisions without further difficulty using the standard Blaise solutions, it became apparent during programming of the **random sampling procedure** that the principle of data storage using Blaise necessitated redesign of the technical concept. The random sample forming the basis for continuous economic computations is generated in the form of a quota random sample. The quotas are based on combinations of the following features:

Household type	Net household income	Earning
Single-person household	less than 1,000 DM	Yes
Marriage/common-law marriages, no children	1,000 – 2,500 DM	
Single-parent family	2,500 – 5,000 DM	No
Marriage/common-law marriage, with children	5,000 – 7,000 DM	
Other household type	7,000 or more DM	

Combination of the feature characteristics produces a maximum of fifty layers, of which 15 are combined with other layers, however, with the result that there are 35 layers to be assigned. The quota plan defines the number of households per layer as a function of population number and structure of the individual states, there being a significant breadth of span: Of the total of 6000 households, for example, 1050 households are located in NRW; the largest layer contains 90 households, whereas there are others with less than five households. The households in each layer must be uniformly distributed within the quarter, in order to avoid seasonal "clumping" effects. Households with similar classification features may be allocated by way of substitution in accordance with a detailed depiction specification if the quota plan cannot be completely fulfilled.

It appears an obvious step to create a number of Blaise files for this purpose, in order to be able to control the procedure with a simple algorithm: As soon as a household is allocated to a layer as a result of its structural features, and as soon as this has been assured, a corresponding change is implemented in the other files and must also be stored, for reasons of consistency. At precisely this point, the Blaise file system is unable to provide any support: It is not permissible to have a number of files open simultaneously or reading **and** writing, as is possible without any difficulty in all PC data-bases. Since API interfaces were still to come at the time of Blaise III, but work was, on the other hand, to be performed using the Blaise system, it was necessary to find alternative solutions. Ultimately, it proved possible to convert the complex random-sampling procedure into a sequence of steps which, from the basic "mass", the households, the features of which accorded precisely with the sorting structure, were allocated for the random sample. All the households not selected were classified as so-called "standby" households. The random sample is filled out from this stock by way of substitution in a second step. Following this allocation the households are subdivided to the so-called "monthly waves" on the basis of the quota plan. Records document these procedures in great detail, with the result that the specialist statisticians can obtain a precise overview of fulfillment of the quota specifications at any time. Administration of the participating households is an essential component of **survey management**. The households cooperate voluntarily in this comprehensive statistical exercise. Careful upkeep of the data

stocks thus obtained is equally significant for the continuity of the reporting group and for rapid and flexible reaction in case of queries, problems and proposals for changes. A number of tasks therefore arise for survey management:

- The addresses must be up-to-date;
- The documentation for the participating households must be prepared and posted;
- It must be possible to directly register any households which wish to take a break for a quote, keep the household ledger in another month, or leave the survey, and to draw corresponding consequences for posting of documents, etc.;
- Where a household has submitted its documentation, a corresponding note of receipt must be implemented, in order that unnecessary inquiries are avoided;
- It should, on the other hand, be possible to send reminders for documents not yet received.
- The participating households receive a modest payment. It must be possible to pay out these amounts immediately and by means of non-receipted transactions;
- It must be possible to draft an up-to-date status report on the status of preparation at any time. A range of business statistics need to be implemented for this purpose.

The backbone for all these activities is the so-called "Household file" (Figure 1: LWRHHD), which constitutes the central control element. In addition, it is also automatically used in the background for other important working operations within continuous economic computations. The data-model fields necessary for this purpose are kept hidden from the user in the dialogue.

The great extent to which flexibility is required of survey management can be illustrated using the example of payment of the premium. It was necessary, since each bureau of statistics wishes to set different background conditions for the payment, i.e., make payment of the payment dependant on receipt of data and processing status, to program a dynamically configurable check condition:

Receipt of documentation	Logical link	Processing status
Yes	and	Undergoing processing
		Enquiry
		Editing completed
No	or	Budgeting completed
		Annual accounting completed
		Forwarded to Federal Bureau of Statistics

The corresponding parameters are set in a Blaise file (INIT), read in in the context of a manipulation operation, and control payment of the premiums.

Survey management is concerned not only with cooperation with the households. Working procedures within the bureaus of statistics must also be controlled. This is reflected in the form of staff administration governing the various editing authorizations of the various users: The Supervisor has access to all functions and is therefore the basic "boss" of the operation, whereas the "standard member of staff" can perform only specific operations, such as the input of questionnaires, and elimination of errors, for example. The Bureaus of Statistics also employ so-called "home workers", who are furnished at home with technical bureau equipment and register and edit the household ledgers at home. Work "portions" are generated by means of survey management for this purpose; this operation also includes the effective anonymization of address information necessary for data security reasons.

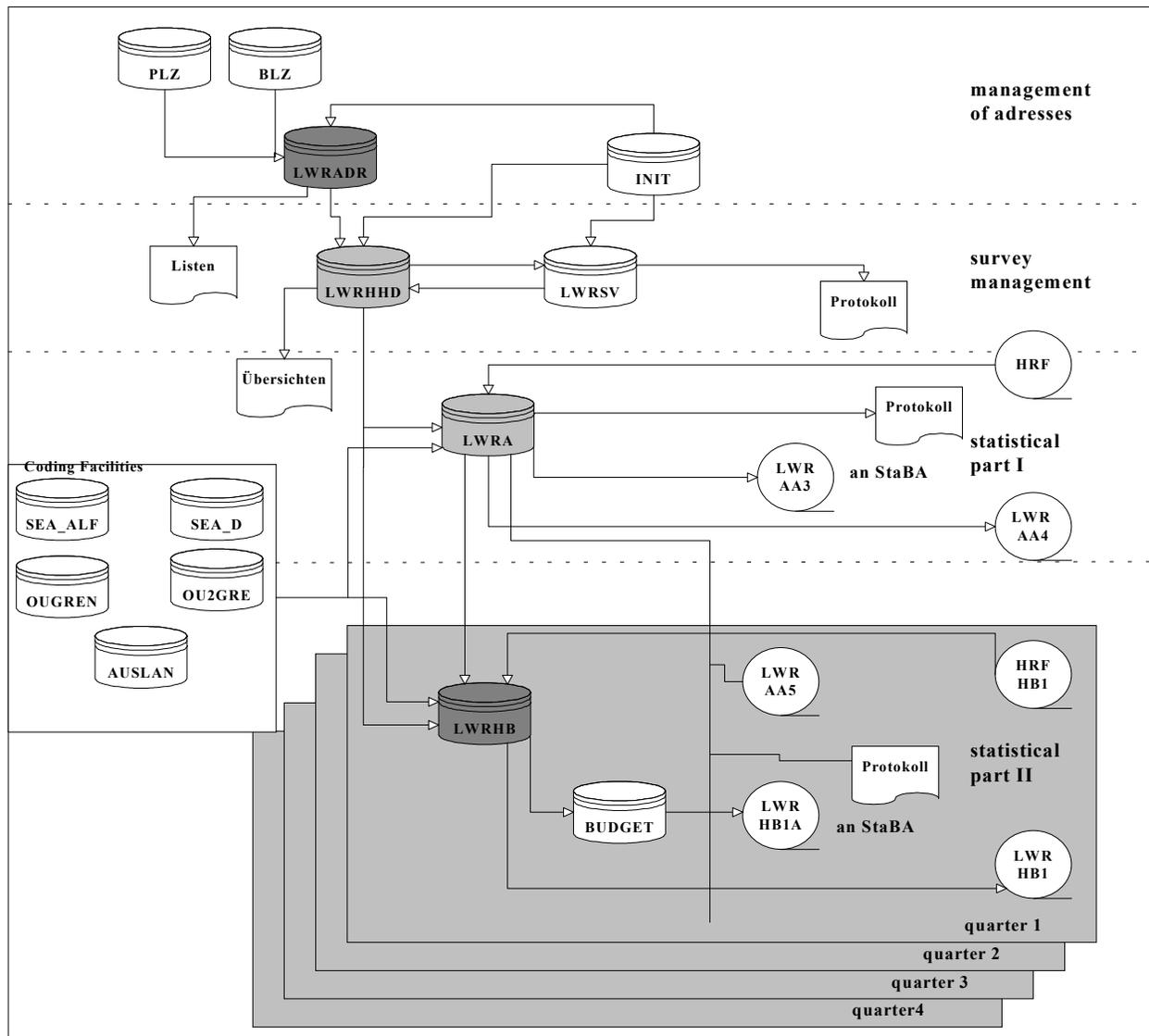


Figure 1: Schematic overview of datamodels in the integrated survey systems.

The central technical element of this application is made up, of course, by the modules for **acquisition and plausibility checking** of the statistical information. Continuous Economic Computations consist of two parts; in the first part, the so-called **General Information**, information on the social status of the members of the household, their employment, information on the size of the household and on net income, domestic situation and availability of consumer goods is obtained. The twelve-page booklet is sent to the households at the beginning of the year and is required back by the end of January.

This questionnaire does not make any special demands in technical terms. Since the list of consumer goods available is modified each year, a flexible access logic which selects and displays the goods concerned in the survey year from the totality of possible goods has been created for this purpose using an *external file*.

The second part of the survey is made up of the so-called **household ledgers**, in which the households record in detail all incomings and expenditure, as well as all changes which have occurred in the household since the last reporting period. Only a few figures are needed to illustrate the statistical and

technical complexity of this section of the survey: The technical department has drafted 200 error specifications in which all fields occurring in practice are involved. Since it is possible to register up to ten persons in each household, and dependency relationships exist between the individual members of the household (parents – children, earners – dependent persons, etc.), the logical necessities and the associated program input can easily be imagined.

The sequencing control of the surveying process itself constitutes an initial challenge:

- Basic socio-demographic features must be extrapolated from the *General Information* element of the survey in such a way that they can be used for plausibility checking of the data in the household ledgers. This operation must be controlled as a function of the data quality achieved and, in addition, it must be possible to repeat it as often as needed and take all quarters into account without endangering data integrity.
- Extrapolation factors applicable for the whole of Germany need to be determined in order to permit evaluation of the data. For this purpose, the technical concept includes central provision by all state bureaus of the plausible data stock for each quarter. The Federal Bureau of Statistics determines the extrapolation factors for the entire random sample and sends it back to the individual states on a case-linked basis. The states, for their part, enter the factors into the data record and are then able to perform evaluations. It must be ensured in the case of this procedure that further modification of the data after export is no longer possible; each and every modification in, for instance, social status would have implications for the extrapolation. For this reason, a concept which "seals" the data, so to speak, at export has been developed and implemented. Such "marked" data records can be viewed only, and not edited in any way. Whereas no system-side solutions for these needs were available in Blaise III, and separate views and browsers needed to be programmed with no small degree of difficulty, it finally became possible to implement an up-to-date access logic system by means of the READ ONLY parameter as from the Windows version (4.3).
- The "sealing" procedure, which provides the greatest possible protection for the data against erroneous editing, has now proven in individual cases to be a hindrance in error elimination. Errors resulting from inadequate specifications were discovered only at the evaluation stage. Thanks to the "sealing" process, however, access to the data had been disabled. It was necessary to introduce a special procedure for cancellation of "sealing" in order to provide a remedy in such problem cases, however. For security reasons, this is available only to the system manager.

The programming of the household ledgers opened up a new dimension to the team of developers: Each data record is made up of more than 7200 fields and takes 35 KB of storage capacity. This volume is the result not only of the extensive person-relevant data on incomings but also, and in particular, of the portion of the household ledgers in which daily expenditure is recorded. Nineteen pages with 600 lines are available to the households for recording in this case, whereby not only the precise designation of the article and the amount are recorded, but also whether the expenditure was in cash or not, in DM or €, in Germany or abroad. An obvious step would be to convert these questions to a table, into which the highly efficient and powerful Blaise coding tools should be integrated. Annoying problems occurred during implementation and test operation, however:

- Since special checks were to be made within the various expenditures – if, for example, a tenant has stated expenditure for rent, this was to be accomplished in a single table spread over 600 lines. The technical people also wanted an error checking system for adherence to upper and lower limits for each expenditure, in addition. Building of the table on the monitor screen was significantly too slow, despite the fact that powerful PCs were available in 1999. Waiting times of up to one minute were not unusual. In order to accelerate the procedure, the range-limit check was firstly redesigned in such a way that it was only applied upon express request. It has now been removed completely. Survey practice also indicates that scarcely more than 300 to 400 entries are normally necessary. The user can now therefore define the number of lines which he intends to enter in a variable before loading the

table. In addition, he can also close the table prematurely using an abort condition (End or Code 9999999, see Figure 2).

LWR 2001 1. Haushaltsbuch Land: 05 HHNR: 1 Prüfung: hard

Drücken Sie die Rücktaste [<-].
Wählen Sie 'Ende - 9999999' um abzubrechen.

Geben Sie einen Text mit höchstens 7 Zeichen ein

	Code	nb	HBAusTag	eu	HBAudtag	Ausl
HB_27[1]	0540351	<input type="checkbox"/>	334,00	<input type="checkbox"/>	334,00	<input type="checkbox"/>
HB_27[2]	011	<input type="checkbox"/>	33,00	<input type="checkbox"/>	33,00	<input type="checkbox"/>
HB_27[3]	0312361	<input type="checkbox"/>	67,00	<input type="checkbox"/>	67,00	<input type="checkbox"/>
HB_27[4]	0942350	x	437,00	e	854,70	<input type="checkbox"/>
HB_27[5]	9999999					
HB_27[6]						
HB_27[7]						

Figure 2: Enter code 9999999 to leave table.

- Allocation of daily expenditure is based on SEA (Systematic Index of Income and Expenditure by Private Households) classification, which is derived from COICOP/HBS systems (Classification of Individual Consumption by Purpose/Household Budget Surveys). It was intended to provide this classification to the user both in the form of a hierarchical "search tree" and in the form of an alphabetically sorted list complete with trigram coding. During conversion of the SEA classification to a Blaise coding file, however, it became apparent that the hierarchical search mechanism does not function correctly under Blaise III and cannot be used. The SEA code contains a two-digit code (01, 02, ... 11, 12) as its first hierarchical level. The order of the codes is reversed (e.g. 02 after code 12) as a result of the coding tool's sorting method, since the leading zero is obviously not taken into account. The SEA classification texts were analyzed, dismantled and re-processed in order to ensure the greatest possible hit rate using alphabetical searching, with the result that they are now available with a short text and an explanatory long text for searching (Figure 3). This procedure has proven extremely worthwhile; approx. 90 % of the codes were found immediately when the goods designation was entered. The hit rate increases by five percent at changeover to trigram mode.² Originally, it was to have been possible to augment the coding files at any time, in order to permit inclusion of new consumer goods and also of synonyms, for example. Since extremely significant performance problems occurred with multiple users in network operation in the necessary read/write mode, a successful change was made to read-only mode. In addition, the users can transfer the coding files to the local machines and access them there directly if the LAN should prove too slow. All the innovations and augmentations for classification have now been centrally compiled and technically revised and sent to the Statistische Verbund in the form of directly usable coding files. LDS NRW has transferred its own application for upkeep of the SEA register to the Statistisches Bundesamt for this purpose.

² It would be extremely practical if the search strings were retained in the entry box at a change of search mode under B4W, as was the case under Blaise III!

Kurztext	Orig_code	Klartext
CD-Brenner	0913015	CD-Brenner, PC-Zubehör
CD-Hüllen	0914030	CD-Hüllen
CD-Player	0911141	CD-Player (ohne solche für Kraftfahrzeuge)
CD-Player Kfz	0911143	CD-Player für Kraftfahrzeuge (Radio-Kassetten- oder Radio-CD-Player-Kombinationen für Kraftfahrzeuge - 0911131)
CD-ROMs	0914013	CD-ROMs, unbespielt, für PC
CD-ROMs	0914024	CD-ROMs, bespielt (ohne System- und Anwendungssoftware)
CD-ROMs	0931012	CD-ROMs, bespielt mit Videospiele
CD-Ständer	0511059	CD-Ständer
CDs	0914013	CDs, unbespielt
CDs	0914021	CDs, bespielt
Cembali	0922130	Cembali
Cervelatwurst	011	Cervelatwurst, Rohwurst
Champagner	021	Champagner (ohne Kantinen- und Gaststättenverzehr)
Champagner	1111057	Verzehr von Champagner in Restaurants, Cafés, Bars, an Imbissständen
Champagner	1112050	Verzehr von Champagner in Kantinen, Mensen

Suchen:

Schlüsseltypen: alfa Seacode tri

Figure. 3: Alphabetical coding in LWR2001 (Table-heading, short text, SEA code, long text)

The methods for **budgeting, classification** of households and **exportation** of ordered data and for **re-importation** of the weighting factors are not examined in more detail here on technical criteria. The technical support necessary for these processing steps has been implemented in comprehensive Manipula programs. Despite the fact that only 1050 households need to be processed in each quarter in NRW, these programs in some cases require extensive running times, due to their technical complexity and the voluminous data models involved. Luckily, the changeover to B4W produced unexpected performance enhancements for data export: These programs are now completed in a few minutes, whereas they needed to run for a number of hours under Blaise III. This is, unfortunately, not also the case for the subsequent plausibility runs necessary (C/S, see below), with the result that dialogue-mode operation is not possible during this period.

4. General aspects

The complexity of the application as a whole can with certainty be illustrated by the fact that more than twenty in some cases extremely voluminous data-models needed to be defined, and just on sixty manipula set-ups generated. The structure chart in Figure 4 also illustrates this. The team of developers managed to model and implement all working procedures in technical programming terms in such a way that they can be started within the application. In addition, the changeover from Blaise III to B4W permitted the integration of the MS Office package. Evaluations and graphic presentations of the business statistics can be called up by the users independently without difficulty. The work input for development of the concept, implementation, tests and introduction was high, however. A total of more than six developer years was invested in all³. Acceptance by users at all the statistical bureaus is good; it has been determined in the course of a long-term observation that all requirements have been implemented in a manner appropriate to everyday practice. Despite this note of acknowledgement, a number of criticisms are nonetheless appropriate on a technical and organizational basis.

³ Particular thanks to B. Wlodkowski, F. Merks, A. Hansen, M. Broose and the Blaise support team from CBS NL.

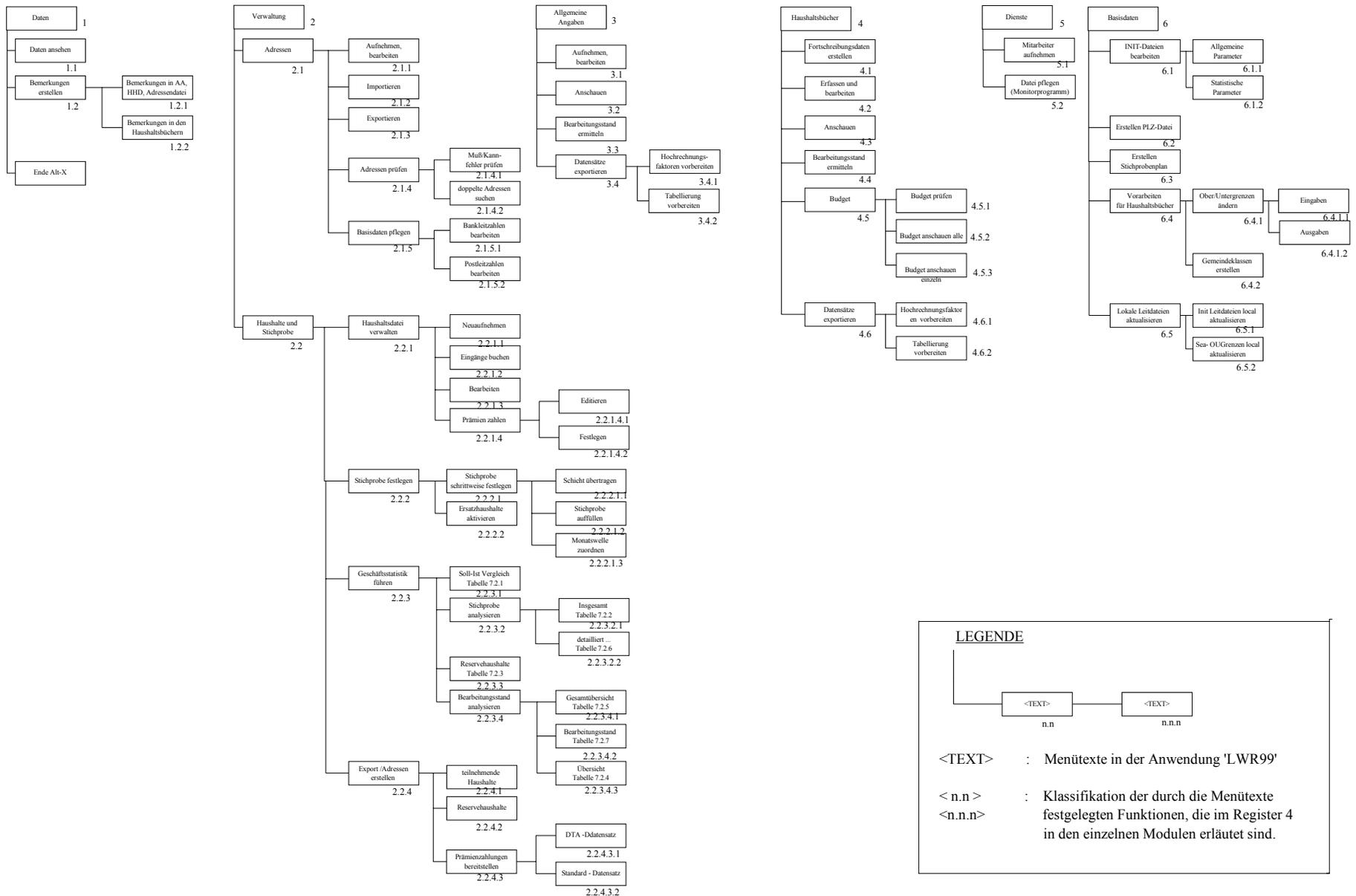


Fig. 4: Functional structure of LWR2001

Blaise's DEP module in principle makes it possible to select various modes for data input and for the corresponding monitor-screen presentation. Only the extremely inconspicuous error counters could be used for error identification in edit mode, however, since development started at the relevant time using Blaise III. Mandatory errors cannot be suppressed in purely interview mode to permit their subsequent correction after enquiry, however. It was necessary to search for other methods in order to permit rapid input of data in interview mode with complete error signalization and the option of suppressing it. A variable with the *check/signal* (C/S) check specification is pre-assigned in the INIT file mentioned above. The overall routing of the data model is structured in such a way that all checks are integrated into a program sequence which is tripped by the check specification (either "checks" or "signals"); see Figure 5.

```

{Check or signal is active until new command}

IF exini.hb_prufe[q] = muss THEN           {muss: var. in INIT.DBD; muss = hard error}
    CHECK
ELSE
    SIGNAL
ENDIF

IF (nb = EMPTY) AND (eu <> EMPTY) THEN
    exini.jahr >= 2002
    INVOLVING(eu, nb)
    "No EURO-cash before 2002!"
ENDIF

IF ((Code = '2099201') OR (Code = '2099301')) AND (nb = '5') THEN
    eu = EMPTY
    "[S19B*] The "Euro" must not be assigned with "x" or "e" for entry of the number
    of garages and parking spaces rented"
ENDIF

```

Figure. 5: Example of external dynamic checking

In order to be able to work quickly, users generally suppress all errors. Since it is necessary to ensure that only error-free material is used for further purposes, however, a concluding check run, in which all mandatory errors are determined, must be performed. This procedure also makes use of the solution described: "Check" is automatically selected in the INIT file and all records are checked in the batch. Simple error statistics are drafted simultaneously. Unfortunately, these runs block the system for a long time – a period of three to four hours is not uncommon, with the result that they are usually performed overnight.

These performance problems in a number of processes in this application have been criticized by users. Around twelve members of staff normally use this application in NRW. The team is doubled at peak periods, however, with the result that all disruptions have a long-term effect on the data preparation process. Improvements in the running of plausibility checks in large tables and in the handling of transactions in the Blaise running-time system would be extremely welcome here.

A problem unsolved up to now is constituted by occasional losses of data: The application crashes on the local workstation during storage of a data record, and the data input for a household, which may in some cases have taken up to two or three hours of work, is partially and in some cases completely lost. Regrettably, no regularities are discernible, signifying that defect analysis and trouble-shooting are difficult. Since the programs used are obviously syntactically correct and also function in principle

correctly, the Blaise system can apparently be eliminated as the cause of the problem. The LDS has made intensive efforts to determine other possible causes of these crashes. The physical structure of the network was first examined. The problems continued to occur even after relocation to a new, and technically optimally equipped, office building. The suggestions provided by CBS of modifying network parameters also failed to produce any radical improvement. At present, attempts are being made by means of a precise addressing system for each PC used to identify possible local features of specific machines as the causal element.

A concluding remark relates to the change of version from Blaise III to B4W. The announcement that the DOS programs would be converted to Windows at the push of a button produced skepticism in advance on the part of the developers. Even the Blaise team was itself a priori convinced that the changeover could be accomplished without difficulty. This skepticism was, it is true, dispelled after the first few successful attempts, but it remains a fact that it is necessary to invest more work than had initially been assumed. This was the result, on the one hand, of improvements which derived from the new potentials (linking to MS Office) and, on the other hand, all the modules had to be re-converted, checked and integrated into the application. During this donkey work, difficulties with the standard conversion routine (B4W, Version 4.2/4.3) for conversion from ASCII to ANSI were just as much of a hindrance as the occasionally incomplete documentation. The support provided by CBS ultimately made it possible to solve the conversion problems well, however.

5. Conclusion

To answer the question initially posed of whether it is possible to draft complex applications for statistical purposes using Blaise, the answer is, of course, "Yes". It has become apparent that it was possible to satisfy the extremely diverse technical requirements practically completely, particularly following conversion to Windows version. The Blaise development environment contains numerous high-power tools for the achievement, on the one hand, of tight intermeshing of computer-based data-processing support and statistical preparation processes while, on the other hand, it is equally possible to implement a survey management system with complete address, household, staff and random-sample administration leaving nothing to be desired on the part of the users. It is therefore indeed worthwhile to further pursue the integrative conception outlined for other statistics, too. It would, however, then be desirable to provide certain modules – such as the address and staff administration modules, for example – i.e., basic "building elements", in a generalized form which can be re-utilized at any time. Blaise's telephone management system is an excellent example of such a procedure. Initial developments in this direction are now becoming apparent in the joint work performed in Germany by the Statistische Verbund; it would be desirable for such modules to be available to all Blaise users – a worthwhile exercise for CBS NL.

References

- Chlumsky, J. and Ehling, M. (1997), Grundzüge des künftigen Konzepts der Wirtschaftsrechnungen der privaten Haushalte. In: *Wirtschaft und Statistik* 7/1997, S. 455-461, Wiesbaden, Statistisches Bundesamt.
- Gertkemper, F. (1997), Ergebnisse der Testerhebung zum neuen Konzept der Laufenden Wirtschaftsrechnungen. In: *Wirtschaft und Statistik* 12/1997, S. 872-878, Wiesbaden, Statistisches Bundesamt.
- Manners, T. (1995), The UK Family Expenditure Survey: comparison of PAPI and CAPI estimates from a controlled trial using Blaise. In: *Essay on Blaise 1993. Proceedings of the Third International Blaise Users Conference*, pp. 101-114, Helsinki, Statistics Finland.

- Manners, T. and Deacon, K. (1997), An Integrated Household Survey in the UK: The Role of Blaise III in an Experimental Development by the Office For National Statistics. In: Actes de la 4è Conférence Internationale des Utilisateurs de Blaise 1997, pp. 197-211, Paris, Institut National de la Statistique et des Etudes Economiques.
- Roessingh, M. and Bethlehem, J. (1993), Trigram coding in the Family Expenditure Survey of th CBS. In: Essay on Blaise 1993. Proceedings of the Second International Blaise Users Conference, pp. 118-132, London, Office of Population Censuses and Surveys.
- Schwamb, J. and Wein, E. (1997), Notes on an Integrated Data Management. In: Newsletter 10, pp. 33-35, ed. by International Blaise User Group, London, Office of Population Censuses and Surveys.

Session 5: Data Editing

- A Blaise Editing System at Westat
Rick Dulaney and Boris Allan, Westat, USA
- Data Entry and Editing of Slovenian Agricultural Census
Pavle Kozjek, Statistical Office of the Republic of Slovenia
- The Role of Error Detection and Correction Systems in Survey Management and Optimisation
Elmar Wein, Federal Statistical Office, Germany

A Blaise Editing System at Westat

Rick Dulaney, Westat

Boris Allan, Westat

Introduction

Editing and delivering survey data pose challenges often quite separate from developing Blaise applications for data collection. Editing often requires that design objectives be modified, interviewer intentions inferred, and data reshaped to our clients' purposes. On longitudinal projects, the editing process must serve two masters: the need to deliver clean data in the current round and the need to field clean and consistent data for subsequent rounds of data collection.

We faced these challenges on a recent longitudinal project at Westat, in which we needed to produce data that accommodated statistical and analytic needs for a particular round of data collection, while processing the data for fielding in a subsequent round. Examination and experience showed that the requirement was to represent the data simultaneously in the Blaise "block" format for continued data collection in Blaise and in a "flat" format suitable for downstream processing on the current round. In addition, updates resulting from either activity had to be represented in both formats. Prior experience had also demonstrated that moving the data quickly out of Blaise into a statistical processing package created cross-platform contentions that made it difficult to iterate between the two databases.

In response, we developed a structured methodology that largely relied on the strength of the datamodel approach in Blaise for maintaining the structural integrity of the data during editing, and on the native ability in Blaise to move data between datamodels designed to represent the same data for different purposes. The basic systems components are:

- There are collection, holding, editing, and delivery databases (all in Blaise) to help keep each part of the process distinct.
- We use face sheets containing comments entered by interviewers, and an annotated list of structural edits failed when the data were checked in the collection database. These face sheets act as a physical record of decisions made, and actions taken.
- A tracking system to monitor and organize editors' work – the system has a list of those editors who are allowed to work on a particular project. Software produces a transaction record for each change in each editing session, so that we know what changes were made, and by whom.

This paper discusses our overall approach to editing complex survey data, first outlining some key issues and components that are common across many large-scale longitudinal surveys, and then describing the design and implementation of the editing system on a particular project. We first implemented this Blaise editing system during the editing and delivery of the Early Childhood Longitudinal Study - Kindergarten Cohort (ECLS-K), sponsored by the National Center for Education Statistics (NCES)¹. The ECLS-K has two major purposes:

- To provide descriptive data on a nationally representative sample of children as they enter kindergarten, transition into first grade, and progress through fifth grade.
- To provide a rich data set that will enable researchers to analyze how a wide range of family, school, community and individual variables affect early success in school

¹ <http://www.nces.ed.gov/ecls/>

This editing model is now being used by other studies at Westat using Blaise data collection.

Editing Considerations

We use the term "edit" to refer to the review and update of survey data. The project context for editing may involve many considerations; those that we identified as key considerations on ECLS-K include:

Design vs. analytic requirements. In some cases, conscious decisions are made during the instrument design phase to accept inconsistent or ambiguous data rather than to challenge the respondent. However, the delivery plan often requires unambiguous variables for analysis, many of which may be derived from other variables. Examples abound in household enumeration; for example, the analytic requirement for father's level of education requires special editing rules in the case of multiple-father households. Or the Round 2 interviewer may have deleted the biological father entered in Round 1 and entered another biological father -- are they the same person? Which is correct? A second consideration arises when the field encounters situations that were unforeseen during the design and which therefore need to be accommodated during the editing process before the data can be made ready for analysis.

Data quality issues. Most survey systems, including Blaise, allow interviewers to make free-form comments analogous to the marginalia recorded on hard copy instruments. These remarks often contain information not represented in the actual survey data, and which may have an impact on the quality of the data, in the sense that survey data should be updated to reflect the remarks. In this sense also, coding open-ended or "other-specify" data can be considered as data quality issues, because again survey data are generally updated to reflect the content of these text fields. In some surveys, this coding operation may dominate other editing activities.

Structural considerations. A Blaise datamodel used as an instrument is normally optimized for data collection, to take advantage of Blaise system features for performing complex flow. Perhaps the most obvious element here is the use of blocks to modularize the instrument and to instantiate different levels of analysis within the instrument. For instance, a household-level questionnaire that includes an enumeration will normally be implemented using one or more blocks at the household level and at least one block at the person level, with each instance representing one household member. However, many analytic files are flat or "rectangular," with one record per household. Person-level data in this approach is repeated for each person on this single record. A successful editing and delivery system will make this transition as seamless as possible.

Timing issues. On longitudinal studies, the editing process is often subject to two constraints: the need to deliver and the need to re-field. If the editing operation discovers data that need updating, but too late to field corrected data in the next round, several problems may result. At the very least, the activity is inefficient, because the same updates will need to be applied after two rounds in separate editing operations. In addition, the incorrect data in the field may affect data collection so that either the wrong data are collected or the data are wrongly collected. Finally, making the same corrections multiple times introduces opportunities for error and therefore the need for reconciliation.

Editing Considerations on the ECLS-K

The ECLS-K began in fall 1998 by visiting approximately 1000 schools and sampling over 20,000 children. Each of these consenting children received an assessment using CAPI, and each cooperating household completed a parent interview conducted over the phone by a field interviewer using CAPI.

Westat conducted subsequent rounds in spring 1999, fall 1999, and spring 2000. Well over 100,000 CAPI interviews have been conducted so far over the four rounds.

The child assessment data is relatively straightforward; after a few screening questions, the child is administered reading, math, and science assessments. Each assessment generally includes a routing section, followed by a high, medium or low version of the assessment. Within the routing section and the assessment itself, there are few skips and no loops. The parent interview, however, is another matter; there is a household enumeration, several sections that are asked twice if twin children are sampled, and various miscellaneous sections that have individual data requirements. The parent interview generally ran from 45-60 minutes. (For completeness' sake, the ECLS-K also collects teacher- and school-level instruments, but these are administered using hard copy.)

After data collection, the data are coded, edited, and simultaneously prepared for re-fielding and extracted for delivery. On the delivery side, the data are weighted, imputed, and masked for public release. In addition, approximately 125 composite variables are created each round from the raw survey data and perhaps other sources such as the frame. This latter is quite important; most of the analytic requirements of the file surface during the development and construction of the composite variables. In many cases, the needs of the composites led to edits that supplemented the checks in the original datamodel. In other cases, the composite specifications supposed that the data were unambiguous; as discussed above, there were many exceptions. Finally, sometimes other activities, such as the coding, yielded data that was now inconsistent for analytic purposes with other survey data; the process of building the composites normally brought these inconsistencies to light.

Based on our experience with other studies and the ECLS-K pilot, we identified a sequence of editing steps for the survey:

- **Comments review**
Comments entered by interviewers during the course of the interview are extracted, converted to an Access database, reports are produced, and any necessary changes are made to interview data.
- **Resolving field problems**
Field staff (interviewers and supervisors) report issues to the ECLS-K Help Desk, and in some cases this means changes have to be made to the interview after it has arrived at the home office.
- **Other-specify coding**
The text strings are reviewed to see whether, in fact, the response should have been coded in one of the existing categories, or if a new category should be created.
- **Structural Edits**
We identified and programmed (using Blaise Manipula) several specific edits. The specifications generally came out of one of the previous steps or from the composite variable process.
- **Prepare File for Delivery**
Again using Blaise Manipula, we "rectangularize" each component (child and parent) in a new Blaise datamodel. We then use this rectangular datamodel to export the Blaise data to ASCII

The Design of the Blaise Editing System for ECLS-K

While much of the documentation necessarily focuses on data collection, the Blaise system is equally strong when used for data editing. The foundation of the technical design for Blaise editing in ECLS-K is the identification of datamodels that map to survey processing needs, and development of rules for transitioning from one model to another. Not including the original datamodel used to define the actual data collection, we identified three datamodels: a holding datamodel, an interactive editing datamodel,

client after the start of data collection -- used in the final delivery file. This file is ready to go into SAS (or any other processing package), generally using the "out of the box" conversion routines that ship with Blaise.

Editing Tools

Several of the data processing steps are not actually tools, in the sense that they are not bundled in a programmer-independent way. Nonetheless, I list them here since they are necessary for the editing process to function, and to serve as a checklist for new projects:

- *Manipula code*: Manipula is used to move cases from data collection to holding, from holding to editing, from editing back to holding, and from holding to delivery. The last one is the most significant of these. Also note that these programs set codes that reflect the current status of the case as it moves through the editing process, so that we can report on how many cases are where. A detailed flowchart is given at the end of this paper showing the specific codes and flow for the ECLS-K editing.
- *Structural edits*: A collection of about 30 edits that clear up problems known to cause problems in the downstream processing. The specs are also reproduced at the end of the document.
- *Face sheets*: Face sheets output from the structural edits help guide the editors. We have found it useful to produce one face sheet for each case that needs attention, and include all data problems. A couple of examples are attached at the end of this document.
- *Rules checking*: When cases come back from editing to the holding database, we run a *rules check* that invokes rules of the holding Blaise datamodel and checks that all the data items are consistent with the skip patterns and other rules.

The tools that the ECLS-K editors use include:

1. **Interactive editing**. This is the heart of the system, controlling and monitoring access to cases and invoking Blaise:
 - a. A shell program collects user information and stores it along with the case ID and start time, and invokes Blaise using the editing datamodel for this case.
 - b. The editing takes place with static checking, meaning that skip patterns are not enforced, which allows the editor to move through a case and fix data items in the order most appropriate for the individual case.
 - c. When the editor exits the case, the shell notes and saves the end time, thus allowing us to report actual editing time per case. (Our final data showed us at about 1.25 minutes per case.)
2. **Coding**. ECLS-K coding happens in two ways.
 - If there are not many actual cases to code (for example a particular other-specify with only a few hundred text responses or with few duplicates), we just use the interactive editing system. The editor jumps down to the question, codes the case by entering the numeric category in the question, and the editing text disappears.
 - Alternatively, if there are a large number of cases or if there is a predefined coding process we want to use, then we export the data into an appropriate database management system (we use Access), perform the coding, and import the data into the editing database using Manipula. We have developed a system that uses Cameleon to write the Blaise

metadata (blocks, arrays, variables, etc.) into Access tables, making the interface to other routines fairly straightforward.

3. **Data Viewer.** It is sometimes useful for the editors just to go in and look at a case, so we have a read-only data viewer that provides access to a copy of the data collection database.

Conclusion

Our experience on ECLS-K with editing the CAPI data using Blaise was very positive. We found it quite straightforward to process updates stemming from the editing considerations described above, while maintaining the full relational integrity of the database. An enormous advantage was our ability to iterate during the entire delivery processing cycle. For instance, if a downstream process such as composite creation or item imputation exposed the need for further editing, we were quickly able to perform the edit in Blaise and retransform the data into rectangular files. A second advantage was the flexibility offered by the multiple datamodel structure; when one piece of the system needed to change (if for instance, a new edit or an additional status code was added) the datamodel was updated and the data transformation routines automatically accommodated the changes.

This approach to data editing – implementing a structured methodology using Blaise – has proven effective in other areas as well. At Westat, we are developing variations of this system to edit hard copy data that has been keyed outside the Blaise system, that is, using Blaise on a project purely to edit data. We also expect the newly released Open Blaise Architecture will allow integration of Blaise editing systems more closely with other technologies.

Data entry and editing of Slovenian Agricultural Census

Pavle Kozjek, Statistical Office of the Republic of Slovenia

1. Introduction

Agricultural Census was one of the main actions of the Statistical Office of the Republic of Slovenia (SORS) in the year 2000. Data were collected by field interviewers on traditional paper forms. Scanning or other character recognition techniques were not used, and relatively large amount of data (more than 120 000 forms, each with 828 data fields) was a good reason to develop an efficient solution for data entry and editing in Blaise. Census applications have to meet some requirements (e.g. speed of data entry) that are not so important with some other surveys, but should be obtained for census data processing.

Agricultural census data entry and editing was processed completely in a local area network Windows NT environment, with Blaise 4 Windows as a main supporting system. This became possible with new hardware and software equipment installed in the data entry department at the beginning of the year 2000. The census data entry application was integrated into a new Blaise database management and administration system, developed at SORS for the general needs of high-speed data entry in a LAN Windows NT environment.

2. Data entry

Census data entry was not integrated with data editing. Main reason for that was traditional organization of data processing at SORS, with two specialized departments: one for data entry and another for data editing. Following this division of labour, the census application was made up of two main data models: one for data entry and another for data editing. Editing data model was developed first, checking the data and retrieving all external information necessary to complete the process of administrative data editing.

Data model used for high speed data entry was a "copy" of the data editing application - structure was the same, but without reference files, and only some formal data checking criteria were enforced. This "top down" sequence of development is perhaps unusual, but in this way compatibility and data transfer between entry and editing databases was ensured in a simple way. Data entry screens were adjusted to the needs of high speed data entry - they were designed as much as possible like paper forms, to help data entrists navigate between questions and fields on the screen. Access to data and applications on a server was enabled through the VB user interface (part of the generator for high-speed data entry applications), using login and password procedures. Administration of data entry was based on physical batches of paper forms, each containing about 50 forms. One batch represents one partial Blaise database that was later in the process added to the final database. This administration was based on generated Blaise datamodels and Manipula setups for handling different Blaise databases and ASCII files, using parameters entered through the user interface. Special attention was devoted to the end users: although some new components were used in the census data entry application, it took only a few days before data entrists learned to use it completely.

2.1 Role of the Generator of high speed data entry applications

The generator of high-speed data entry applications (GEntry)¹ is an in-house developed tool, made for two types of end-users:

- Data entry administrator specifies and generates data model (together with all setups necessary to complete the data entry process), implements it in a production environment and finally sends entered data to the archive and further processing.
- Data entrists use generated application to enter and verify data, using the method of double keying.

The generator is based on functionality of Blaise 4 Windows and Visual Basic. It enables non-EDP people specifying survey data models, and supports and organizes high-speed data entry, verification and process administration on a LAN. Of course, generator can only be successful and efficient with some in-house standards and conventions concerning design of paper forms and data models respectively.

With regular statistical surveys, the procedure of preparing and using the data entry application is highly automated. Some "special cases" need to be improved and corrected "manually". With census, the main data model was not generated, but the rest of the data entry process (administration of data and users) was automated like in the other surveys with high-speed data entry, using GEntry production environment.

3. Data editing

Compared to data entry, the data editing application represented another view of the census database. Blaise screen was different (interviewing mode, with basic meta information on the screen), all data editing rules were included and 4 reference files were used to check and to complete the census data: the address register of farms, the code list of farms, the code list of municipalities and the code list of house numbers. A primary key was defined (in data entry application only secondary) to obtain easy multi-user access to data forms, collected in a single Blaise database (more than 500 M). In the final stage some problems with response time appeared, probably also related to relatively large reference files (up to 500 000 records) and a number of criteria (632 checks and signals)². The editing application also provided the possibility of data entry, for some cases where data were not entered by the high-speed application.

During data entry, some subject matter inconsistencies between collected data and SORS registers and code lists were discovered. Most of them were solved during the data editing process.

4. Findings and their importance for future work

In general, data entry and editing application functioned successfully, without serious problems and the job was finished respecting deadlines. Users (data entrists and data checking staff) accepted it very well.

¹ GEntry-Generator of high-speed data entry applications was presented on a 6th International Blaise Users Conference (IBUC) in Cork, Ireland in May 2000. It is used in SORS production since April 2000, and in May 2001 it covers about 80 % of SORS high-speed data entry (70 surveys).

² Repetitions of the same check on a different field are also counted.

This was the first time at SORS that department for high-speed data entry used form-based data entry instead of traditional record-based data entry (which is used with the old mainframe solution). It seems that users have no problems to accept it.

With census application tried to optimize the situation, where (due to specialized staff, or other reasons) high-speed data entry is used as the first step of statistical process, and data editing is the following step. We are aiming to integrate both steps, where it is applicable, but currently we have to deal with two specialized departments. The only exceptions are some important surveys running in a CATI studio.

Comparing to processes of other surveys with traditional high-speed data entry on a mainframe, some important advantages were noticed:

- Complete process (including high-speed data entry) is defined in a single (LAN Windows NT) environment - easier and more efficient process management, maintenance etc.
- Entry and editing processes can be separated or integrated (entering data directly into editing application)
- Less re-structuring: basically the same data model is used for entry and editing; at the same time it provides structuring and metadata for further processing (e.g. tables in a relational database)
- Form-based data entry provides easier and clearer specification of data editing rules (although it sometimes complicates data modeling).

Lessons learned are already included into the project of building SORS general data editing system, running in a LAN Windows NT environment and based on Blaise and VB. Process management is highly automated, and development and maintenance can be (in most cases) defined as set of rules and templates. This is extremely important when you have to deal with a large number of surveys and a very limited number of application developers.

5. Conclusions

The application for agricultural census data entry and editing represents one of the possible solutions how to integrate high-speed data entry into the statistical process in a LAN environment. At SORS we are using it as a reference for surveys, where high-speed data entry is followed by interactive data editing in Blaise. Rational approach to development, implementation and maintenance is highlighted: we are looking for the efficiency of the complete process of a survey, not just a data entry or data editing phase. On the other hand, solutions should be also general, to cover the wide range of different surveys and to include "special cases" in an acceptable way.

Furthermore, data entry and editing applications should provide an important part of metadata for the next steps of data processing. The project of building general base of SORS survey metadata is under way, and we are working together to find the best solutions. In this way we are trying to move step by step towards standardized and integrated statistical survey processing environment.

References

- Keller, W. J., Preparing for a New Era in Statistical Processing: How new technologies and methodologies will affect statistical processes and their organization, Strategic reflection Colloquium on IT for Statistics, Luxembourg, 1999.
- Kozjek, P., Blaise Generator for High Speed Data Entry Applications, 6th International Blaise Users' Conference, Cork, 2000.

- SORS and Statistics Sweden, Feasibility study on the architecture of information systems and related equipment issues, Study implementation and Hardware/Software Specification for Tendering, September 1997.
- Sundgren, B., An Information Systems Architecture for National and International Statistical Organizations, CES/AC.71/1999/4, Meeting on the Management of Statistical Information Technology, Geneva, 15-17 February 1999.

The Role of Error Detection and Correction Systems in Survey Management and Optimisation

Elmar Wein, Federal Statistical Office Germany

Abstract

This contribution contains reflections about the planning, management and optimisation of data editing processes. The focus is set on information needed for the management and optimisation of data editing. The management of data editing requires information about the state of error correction, real time effort and costs. With the help of the information provided, the causes for deviations between the data editing activities planned and those actually performed should be discovered.

In contrast to that the optimisation of data editing sets the focus on the discovery of poor data editing processes and the most effective corrections. To fulfil these tasks the optimisation of data editing particularly needs raw and plausible data and information about the data editing processing.

This information should be delivered by error detection and correction systems – like Blaise.

Keywords

Process management, data editing, management information, error detection and correction systems, management of data editing, optimisation of data editing

1 Introduction

Increasing user demand for statistical results and (continuous) budget cuts sharpen the view for the necessity of survey processes and consumption of resources. Data editing is a primary survey process which consumes up to 40 % of the resources needed for the production of a statistics.¹ This fact induces a need for powerful data editing methods, edp techniques and management methods. While great progress has been achieved in data editing methods and respective software, the development of specific management methods used for data editing has been neglected. Essential for a management of data editing is a solid planning, the ability to "measure" the performance, to compare real (management) data with the planned ones and to manage data editing activities on the basis of the conclusions drawn.

Rapid progress in the area of information technology (IT) and the fact that not all problems can be solved during the performance of data editing require a continuous optimisation of data editing activities and questionnaires. These activities need data which should be provided by error detection and correction systems.

For that reason the aims of this contribution are to describe the specific aspects of the management and optimisation of data editing, data required for the management and optimisation of data editing, and the

requirements set for error detection and correction systems concerning the provision of appropriate real data.

The contents of the following paragraphs have not been tested in surveys, however they have been influenced by the discussion of a German task force which develops guidelines for data editing.

2 Reflections about a management of data editing

2.1 Methodological framework

Essential for the management of data editing are:

- the definition of data quality which sets the aims for data editing activities,
- the way in which data editing activities are organised, and
- criteria and data used for the judgement of the efficiency of data editing activities.

User demands for data quality influence data editing. There is no uniform definition of data quality, but with regard to data editing the focus may be put on the commonly used criteria "timeliness", "accuracy" and "clarity, accessibility".ⁱⁱ Clarity is an important criterion for data editing because it has to provide in many cases information for user-friendly quality indicators.

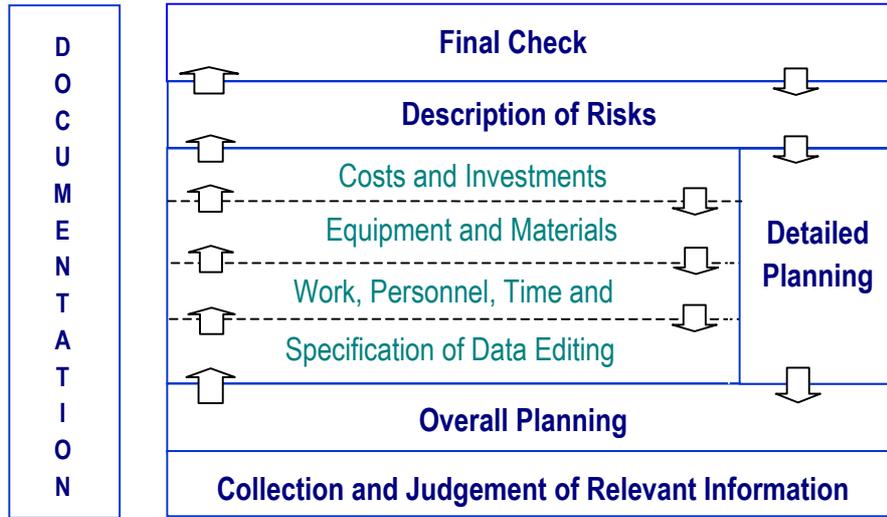
It's a fact that the reorganisation of activities in processes improves the efficiency.ⁱⁱⁱ For that reason data editing activities are combined in data editing processes.^{iv} A data editing process receives raw statistical data and delivers plausible ones as a result of logically connected activities. The design of a data editing process reflects the individual view of an organiser. Data editing processes are designed in such a way as to contribute to the dissemination of statistical results by short **runtimes**, low consumption of resources measured by **process costs** and a user-friendly documentation which is needed for data analysis. They possess only the absolutely necessary interfaces with other survey processes, and complex data editing processes can be divided into logically separated sub-processes. A process owner is defined for every data editing process who needs information, methodological and subject matter knowledge and adequate (IT) equipment.

The achieved "accuracy" of statistical data, needed runtimes and process costs form essential criteria for the management of data editing processes, measured by *real* data of data editing processes and compared with *planned* data coming from the planning of data editing.

2.2 Planning of data editing processes

It is assumed that there are arrangements between users and producers of statistical results concerning their accuracy. They are transformed into operative criteria for the management of data editing processes. Runtimes and process costs are estimated and calculated during the planning of data editing which may be performed in the following steps:^v

Figure: The Flow of the Planning of Data Editing



The figure above shows that the planning of data editing consists of the overall planning with the preceding collection and judgement of relevant information. The main aim of this step is the development of a consistent data editing strategy consisting of data editing sub-processes. They are completely specified during the several steps of the following detailed planning. Specifications of data editing activities e.g. OCR, data capture, (manual) coding and checks represent work to be done. For that reason they form the basis for the estimation of process duration - the basis of process costs. With the help of a powerful meta-data system and relevant project management software, process duration can be estimated for typical data editing sub-processes as demonstrated:^{vi}

The estimation of the time effort for data capture \hat{t}_D is based on the average time needed for capturing an attribute i of a characteristic \hat{t}_{Di} . It is determined by the length of an attribute, their number and the speed of data capture. If more than one attribute is captured, it will be necessary to weight them with their probabilities of occurrence \hat{p}_{Di} and to add the weighted values. If the occurrence of a characteristic c depends on routings the respective estimated probability \hat{p}_c shall be considered. Based on an estimation of the average time needed for checking the captured data per questionnaire \hat{t}_Q and the expected number of questionnaires for data capture \hat{n}_D the basic time effort can be estimated. It should be generally expanded by time needed for non-specific activities e.g. meetings, training. This will be done by a factor of 15 percent so that the time effort for data capture \hat{t}_D can be finally estimated as follows:

$$\hat{t}_D = 1,15 \cdot \hat{n}_D \cdot \left(\left(\sum_{c=1}^C \hat{p}_c \cdot \sum_{i=1}^{I_c} \hat{p}_{Di} \cdot \hat{t}_{Di} \right) + \hat{t}_Q \right)$$

The main advantage of this procedure is the efficient use of existing metadata of the survey contents and assumed sample size and the use of such a formula with real data. Data editing processes which consist of different activities and the time effort needed for computations can't be estimated in the mentioned way; they should be indicated as total numbers. The disadvantage of such an overall

estimation is that there is no opportunity of finding out the causes of deviations during the management of data editing processes.

The calculation of process duration requires the division of estimated time efforts through the general worktime per day/week/month and the available time of personnel or available hours of computation respectively. The result of this calculation should be multiplied with the set of labor costs of the personnel or computations.

2.3 Management of data editing processes

It is assumed that management activities consist of a periodic observation of the error detection and correction and a kind of stock-taking at critical steps / procedures – preferably at milestones. The periodic observation requires an error report which generally contains accounts and graphics on erroneous data and the influence of corrected errors on data. Furthermore a management report should document the performance of data editing by a comparison of the estimated and needed time, the influence on the beginning and termination of all following processes, a comparison between calculated and real costs and an overview of the errors and correction. In addition to that the report should deliver information about causes of time deviations, e.g. differences in numbers of errors and more or less time effort needed for their correction.

This information required for the reports should be delivered from the planning of data editing and measured during the performance of data editing processes. The proposed formula for the estimation of the time effort opens the opportunity to calculate real time efforts. The basis of this calculation is formed by real numbers, which should be provided by error detection and correction systems. They should also be used for the determination of the causes of deviations and should give ideas about how very effective corrections could be performed.

3 Optimisation of survey processes

Changes in the demand for statistical results, problems during the performance of a survey, methodological and technical progress and changing demands for statistical results are the most important reasons for an optimisation of a survey. In this context the focus will be set on the improvement of data editing processes and questionnaire design. Furthermore recent reflections see in the optimisation of survey processes the chance and necessity to improve the knowledge of the personnel who is involved in the survey operations. The aims of an optimisation may be:

1. the improvement of data quality and efficiency of survey processes.^{vii}

Relevant activities are focussed on performed corrections, the introduction of new data editing methods and process redesign. These activities require information about raw and plausible data and documentations of data processes which should contain information about duration and costs.

2. the reduction of burdens on respondents and the prevention of errors.

This aim can be achieved by an optimisation of questionnaires which may lead to a reduction of burdens on respondents and of errors as well. Error statistics may provide useful information for the optimisation of questionnaires.

3. the improvement of the knowledge at a national statistical office.

Knowledge which was acquired during the performance of survey processes, e.g. about the application of data editing methods, may have great influence on methodological guidelines and research which are valuable for colleagues with similar tasks. Error statistics contain important information for methodological research on data editing methods.

The reflections in chapter 2 and 3 demonstrate that the optimisation of data editing sets higher demands on the provision of information.

4 Demands on an error detection and correction system

Error detection and correction systems like Blaise play an important role with respect to the provision of information required for the management and optimisation of data editing. In accordance with the preceding reflections an error detection and correction system should deliver the following information:

Table: Information delivered by an error and detection system

Field	Example	Remarks
Identification number (ID) of a survey	...	This information is needed for comparisons between different surveys.
Name of a file	<i>Stat32</i>	This information is needed for the reconstruction of implausible data during the optimisation of data editing.
ID of a record	<i>112</i>	
Date	<i>04222001</i>	It is assumed that there are general management decisions concerning the work effort for error correction activities. The date is therefore necessary to determine the influence of these decisions.
Time	<i>1503</i>	The beginning of a plausibility check is needed in combination with the beginning of a following check to determine the work effort.
ID of a plausibility check	<i>A025</i>	It is assumed that plausibility checks are combined and integrated in data editing processes. The ID is necessary to calculate the process effort.
Type of correction	<i>manual / automated</i>	This information should be handled like a flag which influences the manner in which time efforts should be calculated.
Corrected characteristic	<i>SocPos[1]</i>	Implausible data can be reconstructed with this information. It is used for the determination of most effective checks.
Original value of the corrected characteristic	<i>2</i>	
Personal ID	...	The personal ID is used for the calculation of process costs.

The proposed information needs some additional remarks:

- The information should be provided only on demand and in separate log files. The further data processing should be performed by a separate management system which should have access to the planned data and formulas as described in the previous sections.
- Data editing non specific activities and general management decisions concerning error correction may heavily influence the distance between two points in time recorded. It is assumed that their influence can be eliminated by the calculation of real time efforts for plausibility checks.

5 Conclusions

Higher user demand for statistical results and data quality and continuous budget cuts lead to new requests for data editing methods and management techniques. The implementation of a process management approach for data editing requires more information about the performance of data editing.

The information should be delivered by error detection and correction systems on demand in a structural form and in separate log files. Especially data about points in time, IDs of plausibility checks, personnel, files, records and original values are needed. On their basis time effort for plausibility checks, process duration and costs may be calculated. A management system should generate management reports on the basis of the information provided.

ⁱ Data Editing Subcommittee of the Federal Committee on Statistical Methodology: "Data Editing in Federal Statistical Agencies", Statistical Policy Working Paper No. 18, Statistical Policy Office, Office of Information and Regulation Affairs, Office of Management and Budget, 1990

ⁱⁱ Yves Franchet (1998). "Verbesserung der Qualität des ESS". DGINS-Konferenz, Stockholm, 18. Bernard Grais (1998). "The Future of European Social Statistics". Mondorf Seminar, pp. 28-31.

ⁱⁱⁱ Bernd W. Wirtz (1996). "Business Process Reengineering – Erfolgsdeterminanten, Probleme und Auswirkungen eines neuen Reorganisationsansatzes". Zeitschrift für betriebswirtschaftliche Forschung, 48, pp. 1023 - 1037

^{iv} Manfred Schulte-Zurhausen (1995). "Organisation". München, 41pp. Günter Schmidt (1999). "Methoden des Prozess-Managements". WiSt, 9, pp. 241-245. Verein Deutscher Ingenieure, Deutsche Gesellschaft für Qualität (1998). "Total Quality Management Prozesse". VDI/DGQ 5505 (Entwurf), Düsseldorf, pp. 2-17

^v Georg A. Winkelhofer (1997). "Methoden für Projektmanagement und Projekte". Berlin, pp. 121-215. Heinrich Keßler, Georg Winkelhofer (1999). "Projektmanagement". Berlin, pp. 162-180.

^{vi} Bundesministerium des Innern (1995). "Handbuch für die Personalbedarfsermittlung". Bonn, B-29 pp. Helmut Wittlage (1995). "Personalbedarfsermittlung". München, pp. 24.

^{vii} Marco Fortini, Mauro Scanu, Marina Signore (2000). "Use of Indicators from data editing for monitoring the quality of the survey process: the Italian information system for survey documentation (SIDI)". Statistical Journal of the United Nations ECE 17 (2000), pp. 25-35

Session 6: Web / CASI

- Some Techniques for Internet Interviewing
Adriaan Hoogendoorn, Vrije Universiteit, Amsterdam, The Netherlands
- Audio-CASI with Challenging Respondents
Rebecca Gatward, Office for National Statistics, United Kingdom
- Dynamic ACASI in the Field: Managing All the Pieces
Boris Allan, Kathleen O'Reagan, and Bryan Lohr
Westat, USA
- Computer-assisted Self-interviewing over the Web: Criteria for Evaluating Survey Software with Reference to Blaise IS
John Flatley, Office for National Statistics, United Kingdom

Some Techniques for Internet Interviewing

Adriaan Hoogendoorn, Vrije Universiteit, Amsterdam¹

Abstract

Internet interviewing poses questionnaire designers for new challenges. This paper discusses a few techniques that we used in the design of a questionnaire on ‘assets and liabilities’ of the CentER Savings Survey. These techniques are evaluated by analyzing log files and respondents’ feed back.

1. Introduction

Data collection over the internet takes place without interviewers. This is one major difference between internet interviewing and the traditional forms of computer assisted interviewing by telephone (CATI) and with personal interviewing where a lap top computer is used (CAPI). The absence of an interviewer places internet interviewing in the field of computer assisted self interviewing (CASI). In internet interviewing the respondent is respondent and interviewer at the same time. The self-interviewer, however, does not get the instructions that CAPI and CATI interviewers usually get. There is no training with respect to the topic of the survey or to the interviewing program. Survey designers have to lay down their efforts in the internet interview program. Technical solutions may compensate for the absence of an interviewer. The absence of an interviewer makes that extra care has to be taken to make sure that questions are clear, and that the respondents are motivated to give valid answers. We will describe and evaluate some techniques that we used in the design of the questionnaire ‘assets and liabilities’ of the CentER Savings Survey in order to achieve these goals.

The CentER Savings Survey (CSS) is a panel survey that started in 1993. Each year, data are collected about income, assets, liabilities, work, pensions, accommodation, mortgages, health, perception of the personal financial situation, income expectations, time preference, interest in financial matters, risk-attitudes, and many other topics. The data of the CSS are collected from over 2000 households in the Netherlands that participate in the *telepanel* of *CentERdata*. The members of the households in the telepanel answer a questionnaire at their personal computers every week. These computers may either be their own computer or a computer provided by CentERdata. The households of the telepanel are recruited by a telephone survey where household sampling is done using the random digit dialing technique, making the panel representative for the Dutch population (see Hoogendoorn, Sikkell and Weerman, 2000). Because the total questionnaire of the CSS is too large to complete in one session the questionnaire is split up into five modules. One of these modules is the questionnaire on ‘assets and liabilities’.

2. Some techniques for questionnaire design for internet interviewing

In our opinion it is important to be restrictive in the information that we use in question texts. In the first place there is a limit to the amount of information that can be put on a computer screen. In the second place we want to reduce the respondents task of reading the question texts by avoiding unnecessary information in question texts. By giving the respondent unneeded information we may encourage him or

¹ The research on this topic was done when the author worked at CentER Applied Research, Tilburg. The author likes to thank Bas Weerman for his expertise on internet interviewing and for providing log files to make this research possible.

her to skip parts of the text, which can result into bad reading of question texts and consequently in giving invalid answers. We want the question texts to be as brief as possible under the condition that the question is clear. This is not an easy task, because respondents are sometimes very different. The questionnaire ‘assets and liabilities’ contains financial terminology where these differences play a part. For some respondents the question ‘Is your mortgage an endowment mortgage?’ is a clear question and their answer is a simple ‘yes’ or ‘no’. Other respondents need to know the characteristics of an ‘endowment mortgage’ before they can give a right answer. We want to keep this extra information hidden as *optional help*, because we do not want to bother those respondents who thought this question was straightforward.

We think it is important that respondents can always *go back* to a previous question in the interview. While a respondent is doing an internet interview he or she may wonder ‘did I understand the previous question well?’, or ‘did I give a correct answer to the previous question?’. Respondents must be allowed to go back to previous questions, either to check the question text and the answer they gave, or to correct a previous answer. Most internet survey software allows the respondent to go back at least one question. We used the Blaise software that allows respondents to go back as far as the beginning of the questionnaire.

In our opinion there are benefits from providing *reviews* during the interview. A review summarizes the answers of a respondent to a part of the questionnaire. It is a form of feedback to the respondent during the interview, and brings a part of the questionnaire to an end. We feel that there is a stimulating effect from such feed back. In addition, a review allows the respondent to check the answers, and can give the respondent the opportunity to change (one of) the answers. Figure 1 shows a computer screen that provides a review of the answers of a respondent with respect to the checking accounts.

Figure 1. A screen shot that shows a review of a respondents’ checking accounts

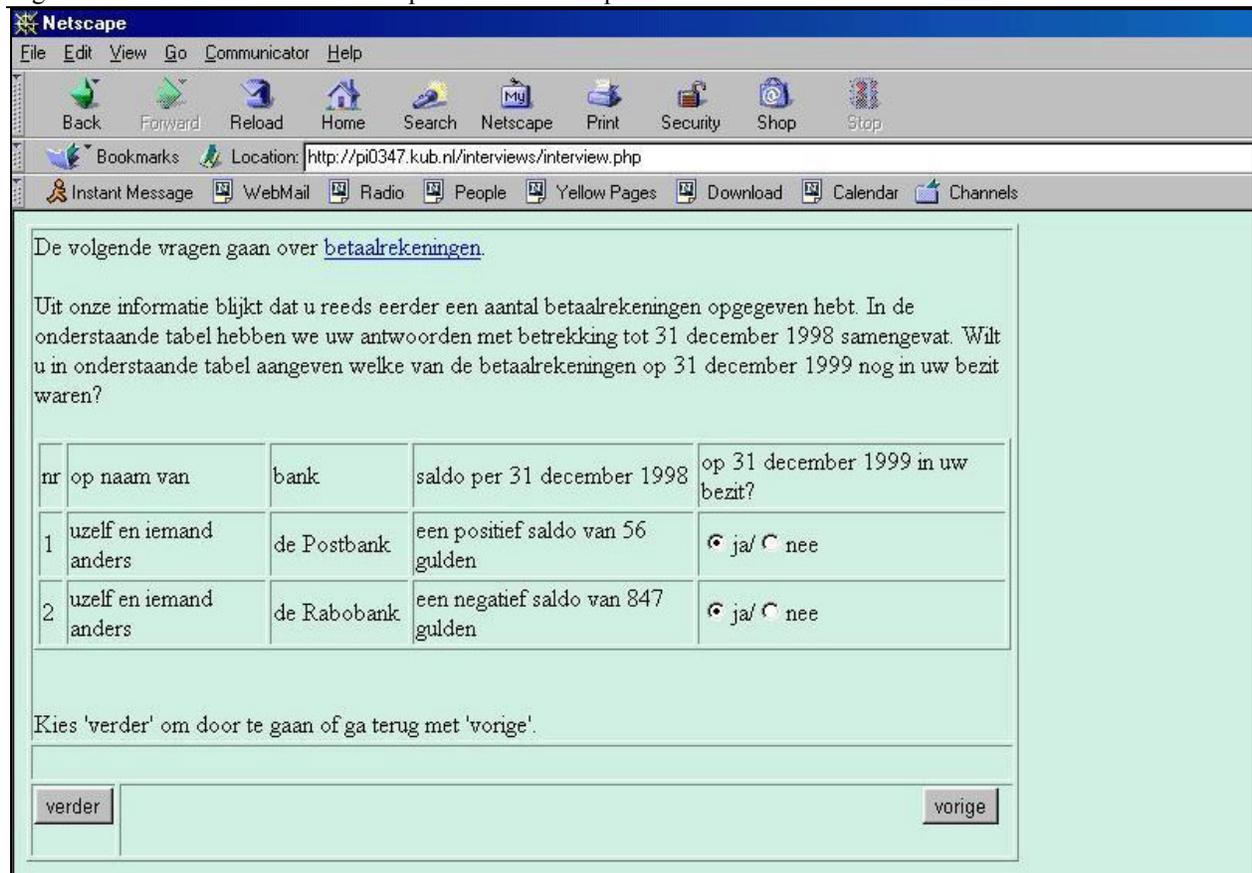


Translation: “You stated that you had 2 checking accounts on 31st December 1999. [table with information about the holder, the bank and the balance of the checking account] Is the information in the table correct? Choose ‘next’ to go on or ‘previous’ to go back. 1. yes, 2. no”

The CSS is a panel survey where we ask respondents to report on many facts of their financial situation. This is especially true in the questionnaire ‘assets and liabilities’. In such a survey it seems efficient to use

preloading. Preloading means that for those respondents that responded in an earlier wave, we present previous answers and ask for changes. In the questionnaire ‘assets and liabilities’ respondents are asked to report on many financial products (‘checking accounts’, ‘saving accounts’, ‘stocks and shares’). Although the balance of checking and saving accounts is likely to change from year to year, most other characteristics (‘Who is the holder of the account?’, ‘With which bank is the account registered?’) are static. Thus, the motivation for preloading is two-fold. First, it is hoped that preloading will make the task of respondents easier, because in many cases the respondent merely indicates if something has changed with respect to last year, rather than to reconstruct complete answers. Second, preloading may make the response more reliable, since last year’s answers form a frame work for this year’s response. Figure 2 shows a screen shot of questions where a respondent is asked whether he or she still owns checking accounts that were reported last year. The appendix contains some information of how this question was programmed to do the data collection on the internet.

Figure 2. A screen shot that shows a question related to preloaded information



Translation: The next questions are about checking accounts. According to our information you reported a number of checking accounts before. In the table below we summarized your answers with respect to December 31, 1998. We like you to indicate in the table below which of the checking accounts were still in your possession at December 31, 1999. [table with information about the holder, the bank, the balance of the checking account and a question whether the checking account is still in possession at December 31, 1999]. Choose ‘next’ to go on or ‘previous’ to go back.

3. Ways to evaluate the interviewing process

The main question in evaluating the techniques mentioned in section 2 (optional help, back options, reviews and preloading) would be: ‘do these techniques improve the quality of the data?’. This is,

the different asset components that the respondent reported last year. It took the respondent 101 seconds to check and maybe correct the answer – this is not clear from the information that we have. The third question, named *CheckLastYearsCheckingAccounts* that we saw in figure 2, is preloaded with the two checking accounts that were reported last year. For each checking account the preloaded information consists of the holder of the account, the name of the bank, the balance of the account and a ‘1’ indicating that the checking account still exists. If the respondent would have answered that at December 31, 1999 the checking account was not in his possession, then the ‘1’ would have been replaced by a ‘0’. Record 4 shows that *TotalNumberOfCheckingAccounts* took the respondent 28 seconds. For the question *CheckingAccount[1].AskChanges* in record 5, we know from the way the questionnaire is programmed that the answer is preloaded with the information of the holder of the account, the bank and the balance of the first checking account, each followed by an indicator that allows the respondent to change these values. The information is presented in such a way that by default it is assumed that only the balance will be changed. In record 5, we see that the respondent did not answer *CheckingAccount[1].AskChanges*, but made a request for the previous question *TotalNumberOfCheckingAccounts* instead. In record 6 we see that the respondent moved on again to *CheckingAccount[1].AskChanges* without changing the answer for *TotalNumberOfCheckingAccounts*. Record 7 shows that the respondent accepted the suggested default where he or she only needs to change the balance of the checking account. Records 8 to 11 deal with the balance of the first checking account. In record 8 we see that the respondent reported that the balance of the first checking account is positive. We see in record 9 that the respondent initially claimed not to know the balance of the first checking account, but that the question *CheckingAccount[1].BalanceCategory*, where the respondent is supposed to specify a category (bracket) for the balance, stimulated the respondent to go back, and after more than five minutes, the balance of the first checking account is set to 4633 euro.

The example in table 1 illustrates the richness of the information in the *log files*. This information poses many questions. Why did the respondent go back when he or she was confronted with the question *CheckingAccount[1].AskChanges*? Did the order in which the questions were asked confuse the respondent? It also seemed that the question where the respondent was asked to specify a category for the balance of the checking account stimulated the respondent to go back and give an exact answer to the previous question. This is an effect of the bracket question we had not expected. It seems as if the bracket question stimulated the respondent to spend 5 minutes on finding the right information. Is this a one time occurrence, or is this a general effect of bracket questions? In this paper, however, we will not answer the questions mentioned here. They are topics for future research. We will use the information in the log files to evaluate the four techniques that we presented in section 2.

A second source of information that we will use to evaluate the use of preloading is a set of five evaluation questions. Every questionnaire of the CentER Savings Survey ends with a brief evaluation by the respondent of a number of salient characteristics of the questionnaire². Respondents are asked to grade (on a scale between 0 and 10) the following five characteristics:

- how *interesting* did you find the topic?
- how *easy* was it for you to do the interview?
- how *clear* were the questions for you?
- how did you like the *layout* of the questions?
- what do you think of the *length* of the interview?

This set of evaluation questions allows us to include the respondents’ subjective evaluation of the questionnaires to the four techniques we presented earlier.

² In fact, since 1995, every questionnaire that is presented to the panel CentERdata ends with these five evaluation questions.

4. Results

We computed some elementary statistics on the basis of the *log files* in order to get some insight into the interviewing process. Our focus will be to evaluate the techniques that we presented in section 2. We analyzed those records in the log files that correspond with the 1583 completed interviews of the questionnaire ‘assets and liabilities’ of the CSS. Those 1583 interviews generated 125,151 records in the log files. Each record in the log file corresponds to a ‘click’, i.e. a request for a next or previous question or for help on a certain topic (see table 2). The 125,151 records correspond to an average of 79 of such requests per interview.

Table 2. Types of requests (‘clicks’) in 1583 completed interviews of the questionnaire ‘assets and liabilities’

<i>type of request</i>	<i>n</i>	<i>%</i>
Next question	119799	95.7
Help topic	1026	0.8
Previous question	4326	3.5
<i>total</i>	125151	100.0

4.1 Optional help on certain topics

Of the 125,151 clicks that respondents made, there were 1026 requests for help on a certain topic. Roughly a quarter of the respondents seemed to use the help option. From the information in the log files, we can investigate for which topics help was requested. The help topics in the questionnaire ‘assets and liabilities’ usually are a description of the asset or liability component. Table 3 shows the number of requests for help on the topic, the percentage of respondents that requested help on the topic and the percentage of respondents that own the asset component.

Table 3. Topics that requested help in 1583 interviews

<i>topic</i>	<i>n</i>	<i>respondent help requests</i>		<i>respondent assets</i>	
		<i>%</i>	<i>%</i>		
checking accounts	163	9.3	90.9		
savings or deposit accounts	85	4.5	66.2		
deposit books	81	4.7	12.6		
endowment insurance policies	76	4.2	10.3		
growth funds	71	4.4	5.4		
employer-sponsored savings plan	60	3.6	58.1		
savings certificates	57	3.1	2.7		
premium savings arrangements	55	3.2	13.2		
put options	50	2.8	1.1		
real estate	47	2.7	6.9		
mutual funds	44	2.3	18.6		
private loans	43	2.4	6.0		
single-premium insurance policies or annuities	34	2.1	17.5		
call options	26	1.6	2.0		
extended lines of credit	25	1.6	16.9		
shares	24	1.4	11.8		
outstanding debts from a hire-purchase contract	16	0.9	2.3		
mortgage bonds	12	0.7	2.1		
annuity insurance policies	11	0.6	2.2		
<i>other</i>	60				
<i>total</i>	1026	24.7			

n: number of times that help topic was requested, respondent request %: percentage of respondents that requested the help topic at least once, respondent possession: percentage of respondents that possess item (asset component).

We see in table 3 that help information is requested for many different (almost all) asset components. We find that help information is most requested for the asset components that are common (checking accounts and savings or deposit accounts). Comparing the last two columns in table 3 seems to suggest that there is some correlation between the requests for help and the possession of assets. In this table we also see some topics for which help is required relatively often. For ‘growth funds’ there are more respondents that request help on this topic, than there are respondent that report to have them in their possession. This suggests that the help information on this topic was very useful. Maybe there would have been more respondents that reported to have ‘growth funds’ if it would not have been made clear what ‘growth funds’ exactly are. From this table we can not tell anything about the respondents that did not use the help topics. It is not clear whether this is due to the fact that all financial terminology was clear to them, or that they did not feel comfortable in using the optional help.

4.2 Go back option

In the 1583 completed interviews, there were 4326 requests for previous questions. This is about three requests per interview. Almost sixty percent of the respondents used the back option at least once. We are interested to know if the back option was used to actually change a given answer or if the back option was just used to check a previous answer. In this respect we have to take into account that respondents can use the back options a few times in a row to go back more than one question. We will call a sequence of successive requests for the previous question a ‘back session’. Table 4 shows the number of times the previous option was used successively. It turns out that there are 2303 such ‘back sessions’. In almost 80% of the cases where the back option was used, it was just used to go back one question. In one case a respondent used the back option 58 times in a row!

Table 4. A count of blocks of successive requests to previous questions

number of times	count	%
1 time	1827	79.3
2 times	195	8.5
3 times	84	3.6
4 times	50	2.2
5 to 9 times	94	4.1
10 times or more	53	2.3
total	2303	100.0

Table 5. A count of independent blocks of successive requests to previous questions

number of times	only check	check and change	total	change rel. %
1 time	607	974	1581	62%
2 times	27	141	168	84%
3 times	4	57	61	93%
4 times or more	4	136	140	97%
total	642	1308	1950	67%

To find out how many ‘back sessions’ were used to actually change a previous answer, we restricted ourselves to those ‘back sessions’ that do not overlap with other ‘back sessions’. Of the 2303 ‘back

sessions' there are 1950 so called 'independent back sessions'³. The results are presented in table 5. We find that in two third of the back sessions an actual change was made. The fraction of changes is larger if the number of successive requests to a previous answer is higher. Although at this stage we have not done any research on what the impact of these changes were, we can conclude that the 'go back option' prevented at least 1308 errors (in a total of 1583 interviews).

4.3 Reviews

The internet interview on 'assets and liabilities' contains many reviews. At the end of the series of questions about each asset or liability component, the respondent receives a review of the answers that he or she gave (see figure 1). The respondent is asked to check if the data in the review are correct. If he claims that the data are not correct, then the program will guide him through the related questions and the answers that he gave earlier, so that he may change one or more of the answers that he gave before. From the information in the log files, we can derive how many reviews were given, and in how many cases such a review led to a change in answers. Table 6 shows the results.

Table 6. Total number of reviews and number (and percentage) of reviews that triggered a correction per asset component in 1583 interviews

asset component	total	change	%-change
checking accounts	1621	136	8%
savings and deposit accounts	1188	101	9%
deposit books	227	19	8%
savings certificates	54	7	13%
single-premium annuity insurance policies and annuities	345	63	18%
savings or endowment insurance policies	194	22	11%
growth funds	100	12	12%
mutual funds or mutual fund accounts	352	37	11%
shares	225	36	16%
real estate	125	14	11%
cars	1136	91	8%
motorbikes	67	6	9%
boats	163	10	6%
other assets	75	3	4%
private loans	109	7	6%
extended lines of credit	286	17	6%
equity-based loan	38	1	3%
debts with a mail-order firm	39	4	10%
loans from family or friends	68	1	1%
study loans	102	3	3%
other loans	27	3	11%
total	6541	593	9%

The bottom line of table 6 shows that of the 6541 cases where the respondent received a review of earlier answers, in 593 cases, being 9%, it led to an actual change in a previous answer. The table contains also some information of the number of reviews and corresponding changes for individual asset components. We may conclude that for the more complex products (single-premium annuity insurance policies and

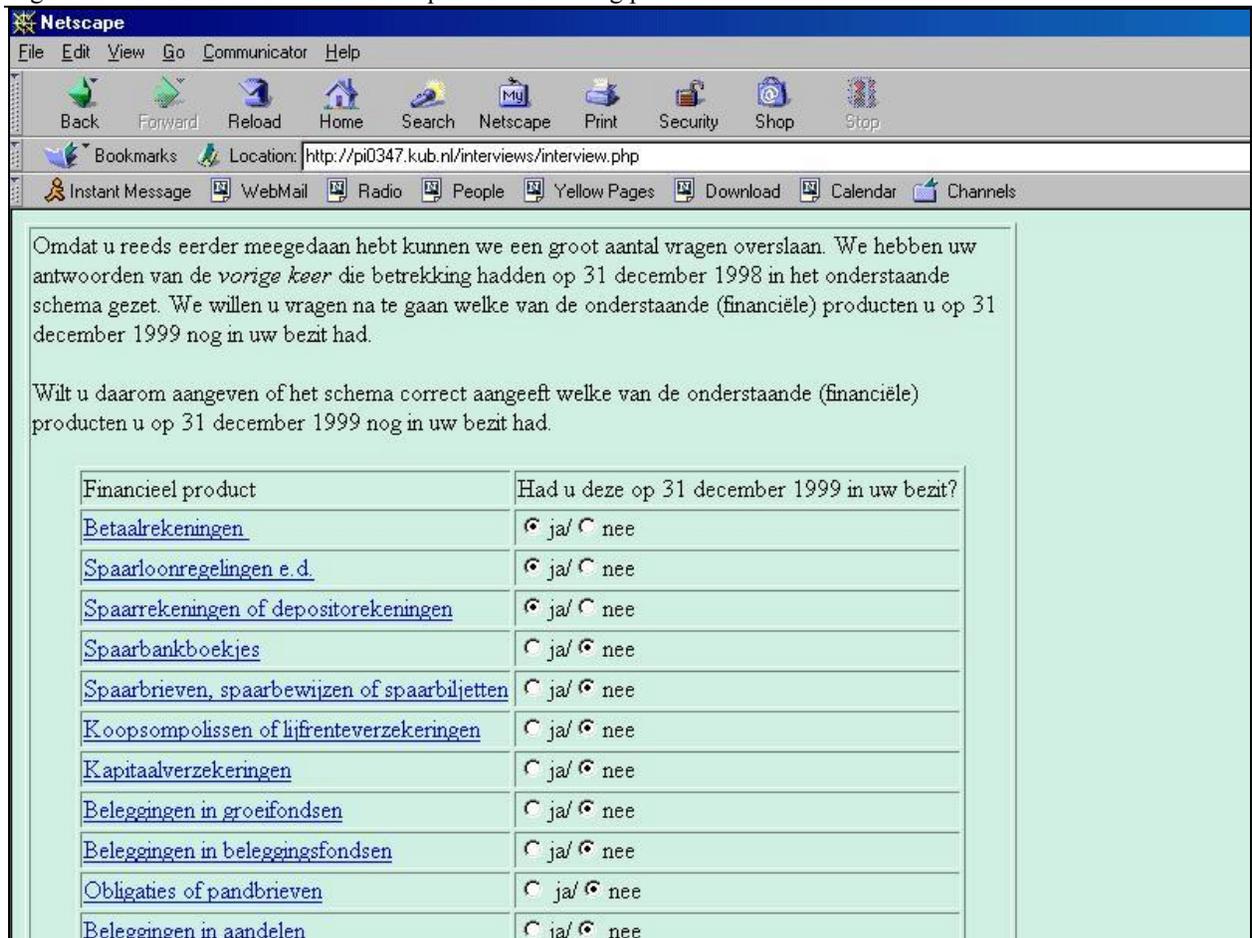
³ An independent back session is a series of successive requests for a previous question followed by a series of the same number of requests for the next question, and where the series of requests for previous questions does not interfere with a series of successive requests for the next question that belong to an other back session.

mutual funds), the rate of changes is even higher than 9%. Although, as in section 4.2, we have no idea of the impact of these changes, we may conclude that the ‘reviews’ prevented 593 errors in 1583 interviews.

4.4 Preloading

In discussing the effects of *preloading*, we will limit ourselves to a few simple questions. One of these questions is: Do respondents that get preloaded data gain time from the fact that they do not have to start from scratch? In fact we expect to gain some efficiency from preloading, since there is a reduction in the number of questions. Respondents that get a questionnaire *without preloading* start with a set of questions to investigate their assets (‘Do you have one or more checking accounts?’, ‘Do you participate in an employer-sponsored savings plan?’, ‘Do you have one or more saving accounts’, etc.) concluded by a review of the asset components the responded claimed to have or not have. Respondents that receive a questionnaire *with preloading* don’t receive the set of questions mentioned above, but start directly with a review of the assets preloaded with the answers of the year before (see figure 3).

Figure 3. A screen shot that shows the question reviewing preloaded information on assets



Translation: Because you responded to this questionnaire before we may skip a large number of questions. We placed your previous answers that refer to December 31, 1998 in the review below. We ask you to check which of the (financial) products you still had in your possession on December 31, 1999. We ask you to indicate in the review below which of the (financial) products you still had in your possession on December 31, 1999. [Table showing ‘financial product’ and the question ‘Did you have these on December 31, 1999 in your possession?’ , Checking accounts: yes/no, Employer-sponsored savings plans: yes/no, etc.]

Of the 1583 completed interviews on the questionnaire ‘assets and liabilities’, 651 respondents received a questionnaire with preloaded information. From the log files we can compute averages on several quantities for the two groups ‘without preloading’ and ‘with preloading’. For both groups we computed the average time spent on the interview, the average number of ‘clicks’ (requests for questions or for optional help) and a few other quantities. The results are summarized in table 7. By comparing the two groups, we have to take into account that ‘with preloading’ and ‘without preloading’ is not the only difference between the groups. The respondents that did not receive a preloaded questionnaire are respondents that did not respond in the last wave (last year), and are in most cases new to the survey.

To our surprise we found no gain in time from preloading, although the number of ‘clicks’ – that correspond to the number of screens shown to the respondent – is 40% less for the group that received a preloaded questionnaire. We found a difference with respect to the use of the help function: the group ‘without preloading’ uses the help options twice as much. This may be explained from the fact that the group ‘without preloading’ consists of many respondents that are new to the survey. The group ‘without preloading’ seems to use the ‘go back option’ more: 3.0 times versus 2.3 times for the group ‘with preloading’, but this difference disappears if we relate this to the total number of clicks. Another surprise is that we find that respondents in the group ‘without preloading’ report on average significantly more asset and liability components than respondents in the group ‘with preloading’. This may be a panel effect (response burden) but this has to be investigated.

Table 7. Averages on some characteristics of the internet interview on ‘assets and liabilities’ for the groups ‘without preloading’ and ‘with preloading’ based on information in the log files

<i>type of request</i>	<i>without preloading</i>	<i>with preloading</i>	<i>signif.</i>
total time spent on interview in seconds	1149.4	1111.4	
number of clicks (requests for next, previous or help)	94.1	58.4	***
number of requests for help on a topic	0.9	0.4	***
number of requests for previous question	3.0	2.3	**
number of asset or liability components reported	4.5	4.2	***
<i>number of interviews</i>	929	650	

Note: ** and *** denote that the difference is significant at the 5% and 1% significance levels, respectively.

An alternative way to evaluate *preloading* is to look if there is a difference in perception of the interview between the two groups. Do respondents that received a preloaded questionnaire perceive the questionnaire different from respondents that started from scratch? To answer this question we will use the set of questions that evaluate the respondents opinion on how interesting, easy and clear they found the questionnaire, and what they thought of the layout and the length of the questionnaire. The respondents grade these ‘dimensions’ of the interview on a scale from 0 to 10. The results are shown in table 8. We find very little differences in the way the two groups of respondent evaluate the interviews. The group ‘with preloading’ found the topic slightly more interesting, but we question if this is a result of selective panel drop out. There appears to be no difference in subjective evaluation of the length of the interview.

Table 8. Averages on evaluations of the internet interview on ‘assets and liabilities’ for the groups ‘without preloading’ and ‘with preloading’ based on information of the evaluation questions

<i>evaluation question</i>	<i>without preloading</i>	<i>with preloading</i>	<i>signif.</i>
how <i>interesting</i> did you find the topic?	7.4	7.6	**
how <i>easy</i> was it for you to do the interview?	7.0	7.0	
how <i>clear</i> were the questions for you?	7.8	7.9	
how did you like the <i>layout</i> of the questions?	7.7	7.8	*
what do you think of the <i>length</i> of the interview?	7.6	7.6	
<i>number of interviews</i>	904	644	

Note: * and ** denote that the difference is significant at the 10% and 5% significance levels, respectively.

5. Conclusion

In designing the questionnaire ‘assets and liabilities’ we used a few techniques to improve the data quality. These techniques are: provide topics that may be unclear to some respondents with optional help, always give respondents the option to go back, give the respondent reviews of answers they gave earlier and use preloading for those respondents that participated in an earlier wave. In this paper we investigated the use of these techniques by analyzing *log files* that monitor the interviewing process, and by *evaluation questions* where respondent give their opinion on aspects of the interview. On the use of optional help on financial topics we found that about a quarter of the respondents used the optional help to find out what a certain financial topic exactly means. We have some indications that the optional help prevented errors. We can also say a few important things about the use of ‘back options’ and ‘reviews’. We found that the ‘go back option’ prevented at least 1308 errors and that the ‘reviews’ prevented 593 errors in a total of 1583 interviews. At this stage we do not know what the exact impact of these changes were to the quality of the data, but the sheer number of errors that can be prevented by these techniques is too impressive to be ignored. On the use of preloading for respondents that responded to an earlier wave in the panel survey we can not say much yet. There appears to be a drastic reduction in number of questions that can be asked, but no reduction in time spent on the interview. With respect to the use of preloading we are eager to continue our research.

References

Bosnjak, M., Tuten, T.L. & Bandilla, W. (2001). Participation in Web Surveys - A Typology. *ZUMA Nachrichten*, 48, 7-17.

Hoogendoorn, A., Sikkel, D. & Weerman, B. (2000), The internet, Blaise and a representative panel, , paper presented at the 6th International Blaise Users Conference, May 2000, Kinsale, Ireland.

Weerman, B. (2001), Internet Interviewing Using Blaise API's, paper (to be) presented at the 7th International Blaise Users Conference, September 2001, Washington, USA.

Appendix

The standard Blaise for Internet Software allows one question per page. In the questionnaire ‘assets and liabilities’, however, we wanted to use tables to ask more than one question at a time. For example, the question *CheckLastYearsCheckingAccounts*, that is shown in the screen shot of figure 2, a table where the respondent is asked to indicate which checking accounts he or she still possesses. In this question we used the ‘B2BList’ procedure. A call to the B2BList procedure is placed in the question text, and parsed by the B2B-software (nowadays the C2B-software) of Bas Weerman – see Weerman (2001) for details – that scans the question text for such procedure calls. The diagram below shows the question text of *CheckLastYearsCheckingAccounts* that contains a call to the B2Blist procedure.

```
CheckLastYearsCheckingAccounts
```

```
"De volgende vragen gaan over ^TxtCheckingAccountsWithHelp.  
@/@/Uit onze informatie blijkt dat u reeds eerder ^TxtOneOrMoreCheckingAccounts  
opgegeven hebt. In de onderstaande tabel hebben we uw antwoorden met betrekking  
tot ^TxtPreviousYear samengevat.
```

```
Wilt u in onderstaande tabel aangeven welke van de betaalrekeningen op  
^TxtThisYear nog in uw bezit waren?
```

```
B2BList(  
  ^NumberOfAccounts2Show, 4, 1112, 0001,  
  '^OldData',  
  'nr', 'op naam van', 'bank',  
  'saldo per ^TxtPreviousYear', 'op ^TxtThisYear in uw bezit?',  
  '1', '2', '3', '4', '5')
```

```
^TxtNextOrBack": STRING;
```

The B2BList procedure must have the following parameters:

- number of rows in the table (r), here the dynamic value of *NumberOfAccounts2Show*;
- number of columns (c), here 4;
- an array of digits (of length c) that denote the type of fields in the column: 1 = a string type, 2 = a yes/no type, here 3 string type followed by a yes/no type;
- an array of digits (of length c) that denote if the fields in the column are asked or shown: 0 = show, 1 = ask, here: 3 show types and an ask type;
- preload string, see below;
- $c+1$ column headers;
- r row headers.

A question that contains a B2BList procedure is of type string, and can be preloaded using the fifth parameter in the call of the B2BList procedure. The diagram below the value of OldData that is necessary to obtain the situation of figure 2. We wrote procedures that obtain the relevant information from the string type question.

```
OldData:= 'uzelf en iemand anders-de Postbank-een positief saldo van 56 gulden-1-'  
+ 'uzelf en iemand anders-de Rabobank-een positief saldo van 847 gulden-1'
```

Audio-CASI with challenging respondents

Rebecca Gatward, Office for National Statistics

Introduction

In March 2001 Social Survey Division (SSD) of the Office for National Statistics (ONS) carried out the first pilot stage of a survey of the development and well-being of children and adolescents looked after by local authorities. Looked after children are either in the care of or accommodated by Local Government Departments. The survey questionnaire includes a substantial set of sensitive questions asked using audio-computer assisted self-interviewing (CASI), the decision to use audio-CASI is explained in more detail later in the 'Background to the survey' section of this paper. This was the first time SSD had used audio-CASI as a method of data collection.

Gaining any young person's co-operation in a survey can be problematic; however, looked after children are more likely than other children to have conduct disorders, emotional problems or hyperactivity. So they form particularly challenging subjects.

This paper will focus on usability and the practical aspects of using audio-CASI with this group of young people. Information for the paper was obtained from interviewers and respondents. Respondents were asked to answer a short set of questions about audio-CASI at the end of their interview. Feedback from interviewers was collected at a face to face debrief session. The paper also describes some of the more practical aspect of developing an audio-CASI instrument and would be of particular interest to others who are planning to use audio-CASI for the first time. The paper also assesses the benefits of using audio-CASI as a mode of interviewing amongst this particular group of young people. Conclusions made in this paper could also be related to other challenging groups of respondents. Other aspects that may also be of interest are that we used standard laptops and that the audio-CASI instrument included some open text questions.

Audio-CASI has been used extensively as a mode of collecting data on sensitive behaviours and this is obviously not the first time the method has been used when interviewing young people. During the development of the audio-CASI instrument we drew on the work of others who have used audio-CASI particularly the work of Jim O'Reilly.

Background to the survey

In 1999, ONS carried out the first national survey of the mental health of children and adolescents in private households in Great Britain (commissioned by the Department of Health, the Scottish Office and the Welsh Office). Interviews were completed about 10,438 children and adolescents aged 5-15 years. This study showed that 1 in 10 children aged 5-15 had a clinically recognisable mental disorder: anxiety, depression, hyperactivity or behavioural problems which had a severe impact on the family. It also showed that children with a mental disorder were three times more likely to have a specific learning difficulty than those without a disorder. In this previous survey the self-completion questionnaire was administered using CASI. For this new survey the questionnaire was developed as an audio-CASI instrument.

Audio-CASI was used as the mode of data collection for two main reasons. Firstly, the sensitive nature of the questions. It is well documented that audio-CASI can increase the reporting of sensitive behaviours. For example Turner et al (1998) reported that respondents were much more likely to report risky behaviours when they were interviewed with audio-CASI measurement technology than when interviewed with more traditional paper self-administered questionnaires. They also state that audio-CASI appears to have a more

pronounced effect on the reporting of behaviours that are particularly sensitive, stigmatised or subject to serious legal sanctions, compared with less sensitive areas of conduct. Although it has been found that CASI and audio-CASI are equally effective at obtaining reports of sensitive behaviours, respondents prefer audio-CASI as a reporting method. (O'Reilly et al, 1994).

The other main reason for using audio-CASI was the nature of the respondent. The young people in this survey sample were more likely to have learning difficulties and problems with concentration. Audio-CASI appears to be a suitable mode for administering surveys to low-literacy respondents, particularly when the survey is collecting personal or sensitive material. (Schneider and Edwards 2000). We hoped that using audio-CASI would enable the young people with learning difficulties to take part fully in the survey and increase the young people's level of attention and interest in the interview.

Not all the young people in this sample had learning difficulties, some had concentration problems and others just wanted to be doing something else or be somewhere else. Therefore it was necessary to develop a questionnaire that could be easily used by the young people who had learning difficulties, without appearing cumbersome or simplistic to respondents who are proficient readers (Couper, 1998).

Survey methodology

The survey involved collecting information about each child from up to three sources: parent, child and teacher. The questionnaire was composed of the following five sections administered using various modes and all contained within one datamodel;

- Parent/Carer face to face interview
- Young person face to face interview (11-15 year olds) (P)
- Young person self-completion section (11-15 year olds) (P)
- Specific learning difficulties test (all children 5-15 years)
- Teacher questionnaire (postal questionnaire, keyed into the datamodel when returned to office).

The two sections, marked with 'P', were set up as parallel fields to allow interviewers flexibility over the order in which they were administered. Depending on the number and complexity of problems the young persons face to face interview could be very long. We felt it was important that interviewers had the option to change to a different section if the child started to lose interest.

Design of the Audio-CASI questionnaire

The young person self-completion questionnaire included questions on the following topics:

- Moods and feelings
- Troublesome behaviour
- Cigarette smoking
- Drinking (alcohol)
- Experience of drugs
- Sexual activity
- Exclusion from school

The questionnaire included both closed and open questions. When developing the questionnaire we set a limit of four response categories for the closed questions. The open questions required only short text or numeric responses, for example: *What word best describes how you have felt in the past 2 weeks?*, up to 20 characters to be entered at this question.

The audio-CASI instrument was developed from a CASI questionnaire that had been used successfully in a previous survey of children. Some less sensitive questions that were in the CASI version were not included in the audio-CASI version but instead asked in the face to face interview. These were mainly questions that had more than four response categories and were part of standard instruments that could not be changed. When designing the audio-CASI instrument we also tried to avoid frequent changes in sets of response categories and to keep the number of different response sets to a minimum. In total the self-completion questionnaire (approximately 130 questions) included 6 different response sets.

The questionnaire was programmed using Blaise 4.2 and administered using interviewer laptops with the standard set up.

Interviewer assisted audio-CASI

The audio-CASI instrument was interviewer assisted. The section started with an introduction read by the interviewer in which they explained how the section worked i.e. they would hear the questions and possible answers through headphones and then enter their own answers into the laptop. Interviewers demonstrated to the child how to enter their answers, how to move on to the next question and how to repeat a question, the young people were also encouraged to ask for help if they needed it.

We hoped that this step by step approach would discourage the young people from being alienated and give them confidence to complete the section. At each stage the interviewer or the child was told via the headphones what they were required to do or happens next.

Changing mode of completion

Ideally we wanted all young people to complete the self-completion section using audio-CASI. However we were aware of the possibility that some young people would be too anxious or have hearing problems. Three possible modes of completion were available; audio CASI, CASI or CAPI (questions read by the interviewer). Instructions and screen layouts were programmed for each of the three modes.

Checking the volume level

Before handing the laptop to the child interviewers checked that the headphones were working and adjusted them to a reasonable volume. The interviewers then passed the headphones to the child who then heard the following instruction: *'The first question you will hear will be a test question just to check that the volume is ok for you'*. This was followed by: *'Is the volume OK for you?'*, if the child said 'no' they were then prompted to ask the interviewer to adjust the volume. The question was then repeated to ensure that the volume was at the correct level.

Practise question

Before starting the actual questions a simple practise question was asked to check that the respondent understood how to enter their responses.

Screen layout

Our aim, when designing the audio-CASI version of the questionnaire, was to keep the screen uncluttered to avoid distracting the respondent. The question text was not displayed on screen because we felt it was more important for the child to concentrate on listening to the question, we felt this was especially important for young people who had reading difficulties. The response categories were displayed in the answer list section of the infopane in the standard way.

Obviously if the section was completed using CASI or as a face to face interview it was necessary to display the question text. To achieve this the colour of the question text changed according to the mode of completion. If audio-CASI was used the text of the question was set to the same as the screen background and therefore could not be read, if CAI or CASI the question text was set to a contrasting colour so it could be read.

The colour switch variable was set up as a parameter in the top-level block of the self-completion section of the questionnaire. Appropriate colours were assigned to 'W' and 'P' in the modelib editor. The parameter (PColour1 and PColour2 – see example below) was switched between '@W' or '@P' depending on the mode of completion.

```
C3G3    "^PColour1  Have you ever used GLUE, GAS OR SOLVENTS?@/  
        PRESS 1 for NO, 2 for YES@/  
        PRESS the WHITE key TO CONTINUE@/ ^PColour1  
        ^PColour2@/ Question G3 ^PColour2@/"  
MML "SOUND (C3G3.WAV)  
      SOUND (Delay.WAV)  
      SOUND (NoYes.WAV)  
      SOUND (White.WAV) "  
: NY
```

Even if the section was completed using audio-CASI it was still necessary for interviewers to be able to recognise the respondents whereabouts in the questionnaire, if they required help. A small identifier was displayed. We were careful not to use an identifier that would be distracting to the respondent such as, the question number or question 1 of 250, so we used the questionnaire variable name.

Labelled keys

The main navigational keys were colour coded, using paper stickers. The <ENTER> key was labelled white and the <F10> key, which had been assigned as the repeat key, was blue. Respondents were also given a card with the following three simple instructions;

- *use the white key to get to the next question*
- *use the blue key to repeat the question*
- *ask the interviewer if you need any help*

When choosing which colours to assign to the two navigational keys we avoided colours that would be confused by people who were colour blind. We also used good quality stickers that would stay attached to the keys.

Audio guidance provided to respondents

Respondents were given audio instructions about how to record their response to a question and to proceed to the next question. For example, if the response categories were simply 'no' and 'yes', they would hear the instruction 'Press 1 for no and 2 for yes' and then 'Press the white key to continue'.

Our aim was to maintain a balance between firstly, providing sufficient guidance and secondly, avoiding frustration for the child by repeating the instructions too often. Instructions were displayed on the first few questions in the questionnaire and then repeated again after each change in response set. If the respondent was expected to record a different type of response such as, typing in their age or text responses they were provided with further instructions. For example, 'Please type in your age in years and then press the white key

to continue'. At these more complicated questions respondents were also reminded to ask the interviewer if they needed help. Other instructions that referred to specific questions were also included.

Recording the audio files

When deciding on the best voice for the audio files, and how to ensure a good quality recording we followed recommendations by O'Reilly (O'Reilly, 1998). A colleague at ONS was chosen as our voice. This meant that they would be easily available to record any additional audio files or make any essential amendments. The person we chose had a neutral and clear voice and did not have a strong regional accent. She also had a great deal of interviewing experience and was familiar with the questionnaire.

The audio files were recorded at a professional recording studio. This meant that the questionnaire text had to be finalised earlier in the questionnaire development stage than usual. In this instance it did not create any difficulties because the majority of the audio-CASI questionnaire was a replication of a CASI questionnaire used in a previous survey. The questions, response sets and instructions were all recorded as separate .WAV files.

Using the recording studio was an efficient method of recording the large number of audio files required (140). The recording process took approximately 90 minutes and cost approximately £200 (about \$300) which included recording time, editing and CD writing.

Other practicalities

Other practical issues considered were:

- Headphones - we purchased reasonably priced, robust and compact headphones.
- Screen wipes - interviewers were provided with screen wipes to use after each interview. These were intended to be used if the young person touched the screen.
- Saving the partially completed interview before passing it on to the child for the audio-CASI section – interviewers were given an on screen reminder to save their interview before passing the laptop over to the child.

Response and length of interview

All twenty-one 11-15 year olds chose to complete the self-completion using audio-CASI and none of the young people who started the audio-CASI section gave up part way through.

The audio section took, on average 25 minutes to complete. The total interview with a young person, including the face to face interview and learning difficulty tests, took on average 110 minutes.

Feedback from respondents

Feedback was collected from the young people via a short set of questions which were administered by the interviewer at the end of the audio-CASI section (a copy of the questions is included as an appendix to this paper). All 21 young people, who completed the audio section, provided feedback.

Overall, feedback from respondents was very positive, all the young people seemed to enjoy completing the audio-CASI questionnaire.

Previous experience of computers

All but one of the young people were frequent users of computers, and all had some previous experience.

General difficulties using audio-CASI

A third of the respondents 'got stuck' whilst completing the questionnaire. In only three instances were the problems related to audio-CASI. The remainder were difficulties interpreting the questions. Problems that did relate to audio-CASI were:

- *'Not much – just the typing',*
- *'I think it's because the computer said press enter and I didn't know how to go back',*
- *'May be when I heard the question I heard it wrong, and forgot what it was, I was listening to the question I was thinking about the answer but forgot what the question was, I think it would have been better if I had listened more carefully'.*

Voice/Volume

The majority of respondents (15) stated that they could hear the questions all of the time, five could hear them most of the time and just one said they could only hear them some of the time. When asked about how well they could understand the person asking the questions, most of the respondents (16) said that they could always understand the person asking the questions, three could hear them most of the time and just two said they could only understand some of the time.

Instructions

Only two respondents felt that the instructions were difficult to follow, the remainder felt they were easy to follow or about right. Respondents were also asked about the frequency of the instructions played after the questions. Four felt they were repeated too often, the majority, 12 respondents, thought they were played at about the right frequency and five respondents thought they were not repeated enough.

General comments about audio-CASI

Respondents were asked to say in their own words how they found their experience of audio-CASI. The most frequent response was 'easy', which was given by ten respondents. Five thought it was 'ok' or 'alright', two had 'no problems', one felt it was 'too easy'. This was despite the fact that a third of respondents had previously stated that they had 'got stuck' at some point during the questionnaire, this implies that the young people felt that overall they had not had any problems.

Respondents were given the opportunity to say whether they had any other problems that they had not already told us about – none of the young people said they had any.

Finally, respondents were asked whether there was anything else they would like to say. Twelve respondents did not add anything. The remaining responses were all positive remarks about aspects of audio-CASI or audio-CASI overall.

- *'it were bril',*
- *'apart from I would like to do it again',*
- *'good it was good fun',*
- *'I wish I had one',*
- *'it's fun',*

- *'I think it is a wicked idea that a computer can speak to you, it is so clever'*,
- *'I liked using it, I love laptops'*,
- *'it's good'*.

Feedback from interviewers

Feedback from interviewers was also very encouraging. All the interviewers agreed that the audio-CASI section was the most concentrated and well-received part of the interview. Interviewers got the impression that the young people really enjoyed completing this section. Often, respondents who were distracted during the face to face interview were really interested and did not fidget or show any signs of frustration whilst completing the audio-CASI section. Interviewers felt that one of the reasons why this section worked so well was because the question text was straightforward and therefore easy to understand.

Interviewers reported that the young people interviewed for this survey did generally have lower concentration levels or were more easily distracted than those they had interviewed for a previous survey of young people. During the face to face interview, which preceded the self-completion section, interviewers sometimes found it difficult to persuade the young people to keep going or had to work hard at maintaining their interest. In some cases interviewers used the audio-CASI section, and the opportunity of using the laptop, as an incentive to keep them going. Interviewers also found it helpful to jump between the different sections of the questionnaire (via the parallel fields).

Interviewers also reported that the whole process of introducing the audio-CASI section to the respondent worked well. Once the interviewer had explained what they were required to do the respondents just *'got on with it'*. There were very few instances when interviewers had to provide assistance to the young person during the completion of the section. Interviewers found that respondents did ask if they required help and if not, interviewers were able to identify when the young person was having difficulties.

According to the interviewers the young people did not have any problems navigating through the questionnaire. One interviewer decided to conduct her own experiment by not labelling the keys on her laptop, instead she told her respondents to press <ENTER> to move on to the next question and the <F10> key to repeat the question again. Her respondents were still easily able to find their way through the questionnaire.

The young people were not tempted to *'play around'* with the laptop once they had come to the end of the audio-CASI section; they just took off their headphones and let the interviewer know as soon as they had finished.

Some respondents seemed to enjoy it so much they asked the interviewers if there was anymore.

Finally, interviewers were asked what they did whilst the young person completed the audio-CASI section. All the interviewers stayed close by in case the young person had difficulties. Some read a newspaper or paperwork, others checked their papers, one interviewer looked at the young person's pets and another talked with the child's carer.

Open text questions

There were very few missing responses to the open text questions. In total the questionnaire included 27 open-ended questions, out of 89 responses there was only one missing answer and this was in response to a question asking why they had been in trouble with the police. The quality of some of the open text responses was poor, mostly due to incorrect spellings. However, only one of the responses was unrecognisable.

Some lessons for the future

Option to toggle between languages

Whilst developing the questionnaire for the pilot stage we found that it was not possible to toggle between playing the audio files and switching them off, depending on the mode of completion. As none of the respondents chose to complete the questionnaire using CAPI or CASI at this pilot stage this did not cause difficulties. However, a method needs to be developed that does not rely on interviewers remembering to plug in the headphones in order to stop the audio-files being heard. Ideally we would like to be able to toggle between languages depending on the mode of completion. The three specific languages would be:

1. ENG (text) with MML (sound)
2. ENG (text) without MML (sound)
3. MML (sound) without ENG (text)

Stop respondents entering early answers

During the pilot respondents were able to interrupt the question by entering early answers. This happened even when the check box 'stop on key' (in the multi-media section of the modelib editor) had not been ticked. If the box is not ticked the respondent should have to listen to the whole question before entering their response.

Checks, signals and error messages

No checks or signals were used in the audio-CASI questionnaire and no guidance was provided to respondents about error messages. Although this did not cause a problem on the small number of cases interviewed for the pilot it is a design issue that needs to be developed further before the mainstage.

Mistake facility

Some guidance needs to be provided to respondents about what to do if they make a mistake. At the pilot stage no instructions were given on how to go back and change their answer. This did cause confusion to one respondent.

Confidentiality

Although lack of confidentiality was not a concern during the pilot we would like to ensure that respondents are not able to go back through the responses the carer/parent gave during their interview. Our solution was to set the section up as a parallel field. This limited navigation to some extent but a more secure method needs to be developed for the mainstage survey.

Conclusion

Feedback received from the respondents and interviewers who took part in the pilot suggests that this particular group of young people enjoyed completing the audio-CASI questionnaire and were able to do so independently. Using audio-CASI as the data collection method enabled us to obtain information from respondents who would normally have refused to continue earlier in the interview.

Our main aim was to design an audio-CASI instrument that did not alienate the respondents and provided those with learning difficulties with the confidence to complete the questionnaire. This was achieved by:

- Providing clear guidance via on screen and audio instructions.
- Maintaining a balance between providing sufficient guidance and becoming too repetitive.
- Introducing the section ‘step by step’.
- Encouraging the respondent to ask questions.
- Minimising the effort required from the respondents.
- Including questions with straightforward wording.

References

Couper, M.P. (1998), ‘The application of cognitive science to computer assisted interviewing. *Cognition and survey research* New York: Wiley, 277-300.

O’Reilly, J.M., et al (1994), ‘Audio and video computer assisted self-interviewing: Preliminary test of new technologies for data collection’ *Journal of official statistics*, 10, 2, pp. 197-214.

O’Reilly, J.M. (1998), ‘Recording audio for audio-CASI instruments’. Unpublished paper

Schneider, S.J. and Edwards, B. (2000), ‘Developing usability guidelines for audiocasi respondents with limited literacy skills’ *Journal of official statistics*, 16, 3, pp. 255-271.

Turner, C.F., et al (1998), ‘Adolescent sexual behaviour, drug use and violence: increased reporting with computer survey technology’ *Science*, May 1998.

Appendix A
Feedback questionnaire for young people

IntroFd INTERVIEWER:
Please explain that we are interested in finding out what they thought of hearing the questions through the headphones and entering the answers on the laptop themselves.

CompUse RUNNING PROMPT
Have you used computers....

1. a lot
2. a bit,
3. or have you never used a computer before?

Probs Did you get stuck at all?

1. Yes
2. No

WhatPrbs *IF PROBS=YES*
Where did you get stuck?
PLEASE OBTAIN AS MUCH DETAIL AS POSSIBLE

Hear RUNNING PROMPT
Could you hear the questions...

1. all of the time
2. most of the time
3. or just some of the time?

Voice RUNNING PROMPT
Could you understand the person asking the questions...

1. all of the time
2. most of the time
3. or just some of the time?

Instr RUNNING PROMPT
Were the instructions....

1. easy to follow
2. about right
3. or difficult follow?

InstrRp

RUNNING PROMPT

Were the instructions that are played at the end of some questions repeated....

1. too often,
2. about right
3. or not enough?

KeyB

How did you find entering your answers into the laptop?

INTERVIEWERS: PLEASE PROBE IF HAD ANY PROBLEMS

AnyOth

Are there any problems that you have not already told me about?

INTERVIEWER: PLEASE PROBE IF HAD ANY PROBLEMS

AnyCom

Is there anything else you would like to say?

Dynamic ACASI in the Field: Managing All the Pieces

Boris Allan, Westat
Kathleen O'Reagan, Westat
Bryan Lohr, Westat

Introduction

While Blaise is an unusually powerful and versatile system, the successful execution of research studies usually requires considerably more than the programming of the Blaise application itself. In particular, a number of technical management processes may be needed in support of data collection and post-collection processing in complex, large-scale surveys. This paper examines the approaches and methods used to manage the complexity of a longitudinal Blaise field data collection involving a mix of CAPI and ACASI interviewing. The sources of complexity in this study included:

- The extensive use of audio-visual enabled computer assisted interviewing (ACASI) with touch screen laptops for three different types of interviews (18 and over, 12-17, and 8-12 years), and the associated need to manage the disparate elements.
- ACASI interviews in two languages (English and Spanish) in the written and sound presentation of questions, question response categories, the selection of advertisements to display or to hear, and help texts.
- The need for confidentiality both in the interview situation and in processing at home office.
- Processing for monthly updates of audiovisual and sound files in both languages that were presented to respondents.
- A daily update of information about the interviews in progress for the helpdesk and for quality control, including the ability to rerun an instrument at the helpdesk to ascertain its current status for assisting interviewers with questions.
- Regular updates of SAS-compatible datasets for statistical analysis and monitoring purposes.
- Longitudinal refielding of cases with appropriate data in subsequent waves of interviewing.

The processes described in this paper were employed for the National Survey of Parents and Youth (NSPY) conducted as part of the Evaluation of the National Youth Anti-Drug Media Campaign by the National Institute on Drug Abuse (NIDA). The sections below discuss the instrumentation components, home office processing, and the management systems developed to support the needs of dynamic ACASI projects.

Blaise, ACASI and Field Management Components

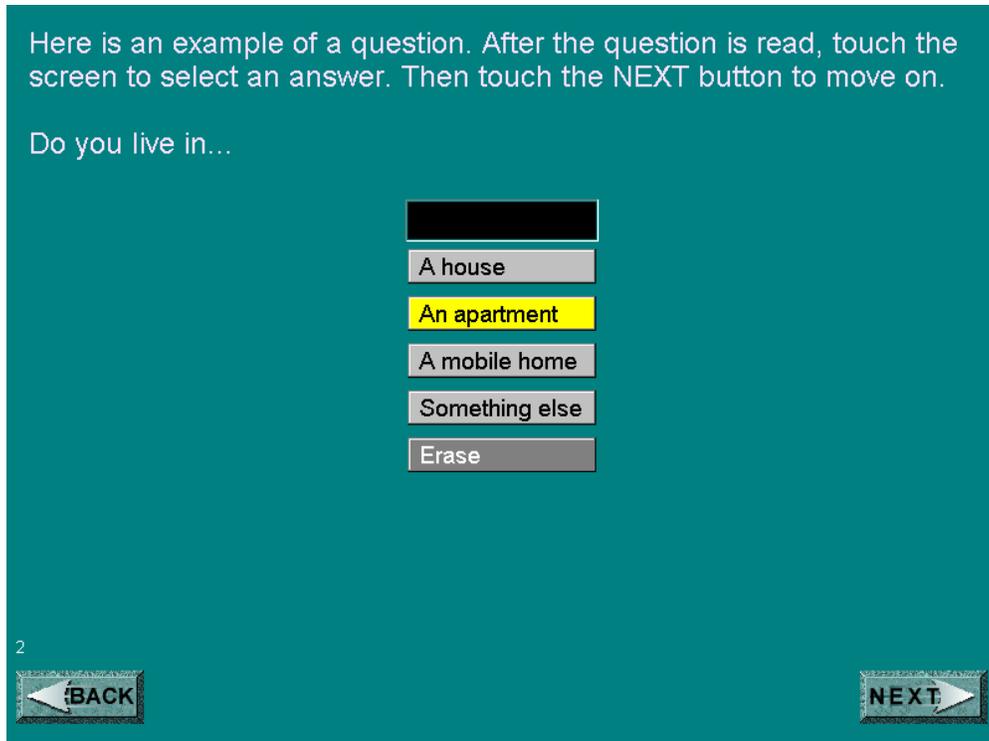
Questionnaires for NSPY were administered in-person in the homes of respondents. Since NSPY was designed to collect sensitive data about drug usage and awareness in the sample population, a certificate of confidentiality was given to each respondent household from the U. S.

Department of Health and Human Services to assure respondents of confidentiality in the reporting of data.

During the course of the interview, the questionnaire screened respondents to determine eligibility and then selected one or two youths from each eligible household and one or two parents, depending upon information provided by respondents, to complete the full interview. The CAPI questionnaires were programmed in Blaise for the CAPI interviewer-administered portions and in Visual Blaise for the audio-visual self-administered ACASI portions.

The survey instruments represented a complex design and data model. The requirement to accommodate both CAPI and ACASI modes for data collection further increased the complexity of the instruments. It was necessary to support Blaise 4.1 and Visual Blaise environments in differing sequences to allow for mixed mode interviewing from CAPI to ACASI. Also, because of the CAPI program's random sampling algorithms to select respondents for the confidential ACASI interviews, there were possible multiple, confidential ACASI interviews with different individuals. It often required multiple visits to the households to complete the ACASI portions with the selected respondent.

Specific sections of the interview that asked sensitive questions and response categories were administered in ACASI mode. For these sections, respondents (parents and youths) used headphones to hear the questions and responses displayed on the screen and used the touch screen to select their response from the answer categories provided. The figure that follows shows an example ACASI screen programmed in Visual Blaise.



This screen is a sample from the tutorial used to instruct respondents on completing the ACASI portions of the interview. The screen displays question text and responses. The respondent would hear a voice reading the question text and responses through the headphones provided. The respondent would select their answer to the question by touching the button for their response choice. An erase button provided the capability for the respondent to erase a response and select a new answer before proceeding to the next screen.

The questionnaires were administered either in English or in Spanish depending on the language requirements of the respondent households. After the text for the questions was developed, the audio files were recorded at a professional recording studio. When question text was changed, the audio files had to be re-recorded in both English and Spanish. Individual .WAV files were recorded for the ACASI instructions to respondents, question text and response categories.

The programming design of these instruments ensured confidentiality of the collected data in both the CAPI and ACASI portions of the survey through a system of "gates." Gates are programmed instructions within the instrument that prevent the back up and review of data to previous questions or sections in the instrument. These programmed gates were located at strategic points within instrument. Once the course of the interview passed a gate, it was not possible to return to an earlier portion of the interview to review questions and collected data, even if the interview had not ended. If the respondent stopped the interview before completing the ACASI portion, the gate securing the data would not be passed, and the data would not be saved.

The interviewers used an Interviewer Management System (IMS) on the laptops to track and view the status of their assignments and enter record of calls information. The Supervisor Management System (SMS) was a web-based application running on a home office server with an Oracle database. When interviewers transmitted to home office, information was exchanged with the SMS regarding case status and assignments.

Home Office Processing

Field procedures and home office processing requirements were defined to ensure the appropriate administration and management of the study in the field and appropriate processing of information after data collection.

The Media Campaign was an ongoing program using mass media venues to run advertisements and other messages to communicate the anti-drug abuse message. These advertisements and messages were run in film, radio, television and print. The Media Campaign used different advertisements and messages based on a schedule for each type of media, and these advertisements changed at different time intervals, depending on the type of media.

Since a goal of NSPY was measuring youth and parent exposure to current Media Campaign messages and tracking the effects of advertising changes, frequent changes were made to the media files. Management tools were needed to support the following operational and analytical tasks:

- The Help Desk had to monitor the progress of interviews on a daily basis for quality control purposes and provide resolutions for identified problems,
- Monthly updates of sound, video, and graphics media files had to be tracked and transmitted to field staff for electronic updating of instruments on field laptops to keep media presentations in the instrument current for data collection, and
- Regular updates of SAS-compatible files were needed for statistical analysis.

Management Tools

Two management tools were developed to meet the needs of the project. A Records Management Tool was developed to automate structural data manipulation at home office; and the Aladin Management Tool was created to provide case level information to the Help Desk, programmers, and data editors.

Records Management Tool

Help Desk and quality control staff required up-to-date information about the status of cases and interviews in order to monitor progress in the field. To meet this requirement, new zip files were transmitted from the field on a daily basis. These files were posted to secure database for processing. The information was then moved to a public database once the identifying information was suppressed.

The Records Management Tool was developed in Maniplus and Manipula to automate the process of managing cases at home office and resolving problems related to Case IDs and records. The Records Management Tool provided access to three Blaise databases to ensure data integrity. The databases included:

- Private Database containing all data received from the field,
- Public Database providing data items needed for most help desk and quality control investigations, and
- Edit Database containing updates and edits posted at home office.

Structural updates to the databases were automated by the Records Management Tool to minimize errors. For example, if the interviewer completed an interview using an incorrect CaseID, it was necessary to move the CAPI and ACASI data to the correct ID and re-initialize the original ID.

Key functionality included:

- Delete Records Function, which allowed for the deletion of a record from any or all of the databases;

- Insert Records Function, which allowed for the insertion of a record from any or all of the databases;
- Extract Address Information Function, which allowed for the extraction of individual or household address information;
- Browse Function, which allowed for browsing of all databases;
- Swap Data Function, which allowed for the swapping of records in any or all databases between IDs; and
- Help Function

Manipula extracts were run weekly on the databases to convert the Blaise files to SAS files for further analysis and QC activities at home office.

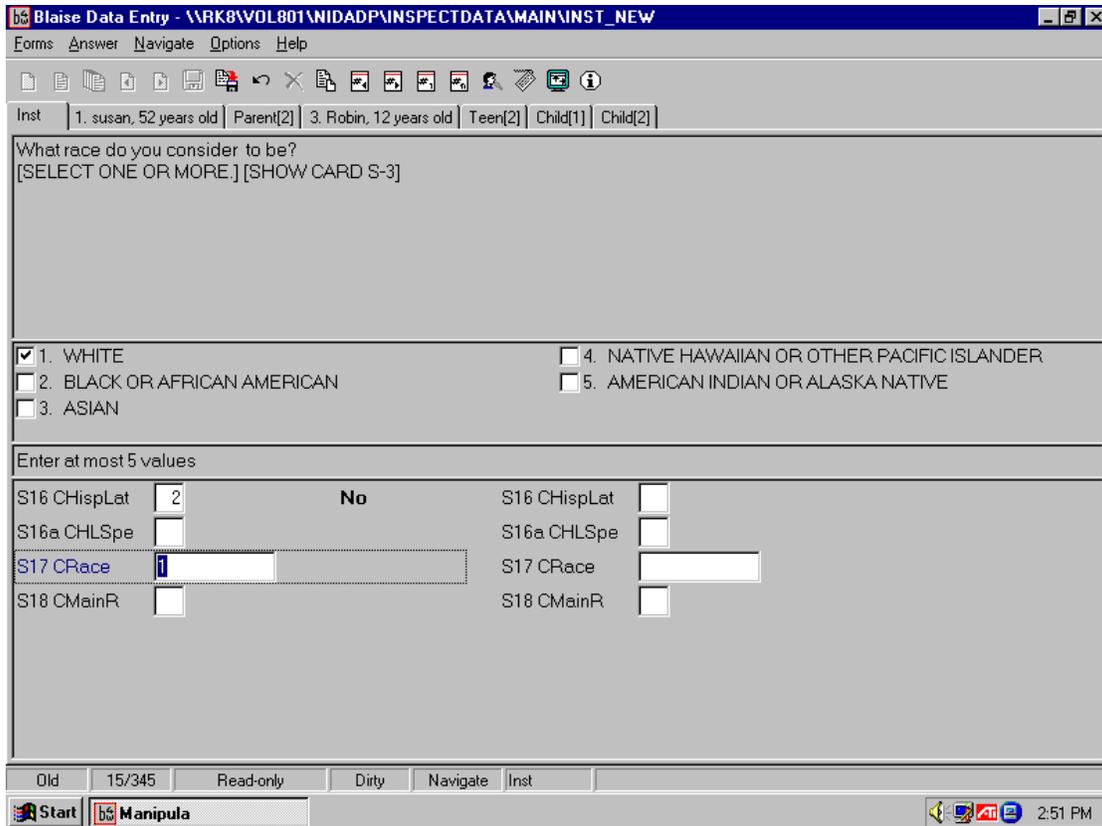
Aladin Management System

The Aladin Management System (an inverted acronym for “NIDA Live Access”) provides an up-to-date controlled environment for central access of NIDA databases and programs. The system was developed in Maniplus and calls Winbatch, Access, HTML, Excel, and Word executables and documents. The system is password-protected to ensure that only authorized staff can access NIDA databases and programs. Aladin provides system management functions to set file properties that protect against the accidental execution of programs and deletion of data.

The Aladin Management System supported multiple home office processing functions including:

- View current wave database,
- Test programs,
- Developer environment, and
- Data editing.

View Current Wave Database: Help Desk staff and others monitored the status of cases and interviewer transmissions and assisted field staff who encountered problems using the capability to view the current database using the Help Desk Viewer. This viewer calls the Blaise DEP running under a Maniplus shell as shown below. The Viewer enabled project staff to review the key fields in the current wave of cases. From the Viewer staff could search for a specific case, run the Blaise DEP, view the audit trail, browse the database, or compare case status information in Oracle and Blaise.

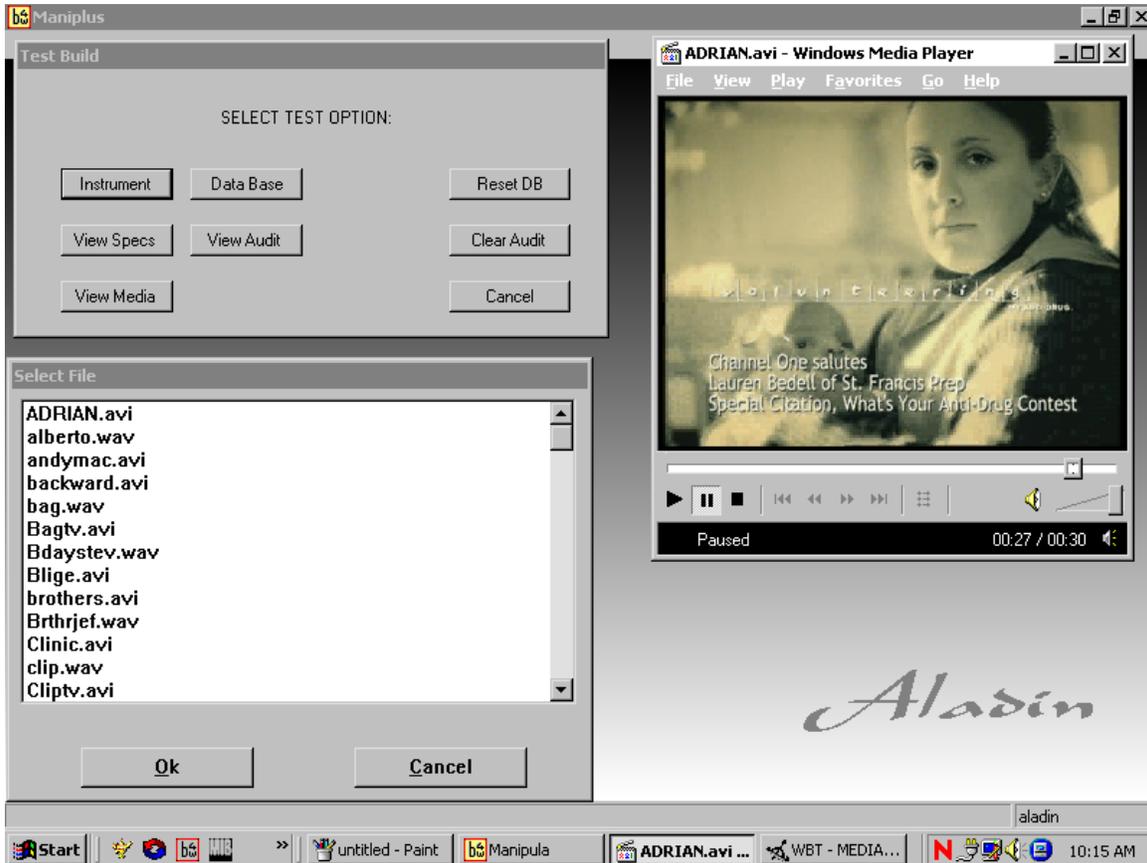


For example, Help Desk staff could search for a specific case by ID and then execute the Blaise DEP for that case. The home office staff could see the specific responses entered by the interviewer even if the case was only partially completed. Since the media presentations were determined by sampling algorithms executed at the time of the interview, this was used to verify the sampling program and determine which ad was played. By using the view audit trail capability, it was possible to see the combined audit trail for the CAPI and ACASI portions of the interview and determine if the interviewer or respondent had particular difficulties.

The browse database capability provided access to the ~.BD files. Home office staff could access the Blaise structure browser to select data fields for viewing. The Oracle/Blaise compare function was used to view case status data from the IMS, SMS, and Blaise interview. Data from the IMS and SMS were stored in Oracle, and the interview data were stored in Blaise. This management function used Microsoft® Access® and required the import of files from the secure Oracle server and text files created by Manipula from the public database.

Test Programs: The Test function enabled developers to perform instrument and database testing, review audit trails created during testing, and review instrument specifications. Aladin's Test function played a key role in ensuring the accurate testing for the monthly updates of the multi-media files (sound, video, and graphics) required to reflect current advertising in the media. The test function enabled developers to view all media files required for an update. The video .AVI and sound .WAV files could be viewed separately through the Windows Media

Player. The instrument had over 3,000 English and Spanish .WAV files to support the CAPI instrument, and over 150 video clips .AVI and .WAV sound clips from radio ads to track and manage.



Developer Environment: This aspect of the system was for systems programmers to refresh the databases and files supporting the system. Batch programs were executed on a daily basis to capture up-to-date information from Oracle containing SMS and IMS information, the Blaise database, and audit trails. Source Integrity was used to manage current versions of the systems code.

Data Editing: During the data collection period, sometimes a help desk report would require that case data already sent from the field to the home office be examined and modified. Once the case was examined through the Blaise DEP or the audit trail viewer and required data changes were identified, the data editing function permitted updates to the database. If appropriate, the updated cases were sent back to the field to complete. In other instances the cases were considered final and ready for analytical processing. Data updates introduced by the data editing function were also captured in the audit trail and indicated as changes made by home office. The system handled re-encryption of the data files as appropriate.

Conclusions

The NIDA study had complex requirements in instrument design, data collection, and home office processing. These project complexities required a systems approach that incorporated diverse elements: Blaise, Manipula, Maniplus, Access, Oracle, and other programs and utilities.

The Blaise program provided the basic platform for the NSPY CAPI instrument, and Visual Blaise enabled the execution of the ACASI portions of these instruments. Additional utilities, like the Record Management Tool and the Aladin Management System provided standard interfaces and controlled environments at the home office for managing development tasks, field monitoring, quality control tasks, data management, and other home office processing tasks. By implementing a controlled environment through these methods and approaches, the project ensured effective data handling, secure data repositories, and standardized methods of development and testing.

Computer-assisted Self-interviewing over the Web: Criteria for Evaluating Survey Software with Reference to Blaise IS

John Flatley, Office for National Statistics

1. Introduction

For those in the statistics business, web technologies offer the potential to speed the process of collecting data; to reduce the costs associated with collection and processing, and to make data accessible more widely and more timely. It is not surprising, therefore, that national statistical institutes (NSIs), such as the Office for National Statistics (ONS), are racing to embrace the Internet to enhance the service they deliver to government, other customers and to citizens. This is the business imperative. In addition, governments have added political impetus. For example, in the United Kingdom (UK) central government has set a target for all public services to be available on-line by 2005. Similar, but more ambitious, targets have been set in Australia (2001) and Canada (2004). As the number with Internet access grows and people become accustomed to using it, there is likely to be increasing demand from the providers and users of official statistics to use the web for such purposes.

Social Survey Division (SSD) of the ONS is examining the feasibility of using the web for the surveys that it undertakes. Nationally representative surveys of the private household population carried out on behalf of the ONS or other government departments, forms the core business of SSD. Such surveys employ random probability sample designs and are voluntary. SSD surveys tend to be undertaken by trained interviewers using computer assisted interviewing (CAI). Computer assisted personal interviewing (CAPI) is often the single mode of data collection. There are examples of surveys that combine CAPI with another mode. For example, the Labour Force Survey (LFS), which is a panel survey, uses CAPI for first interview and offers a computer assisted telephone interviewing (CATI) option for subsequent interviews (there are five interviews in total). On both the Expenditure and Food Survey (EFS) and National Travel Survey (NTS), following an initial CAPI interview, respondents are asked to self-complete a paper diary. SSD is also commissioned to sample other populations, such as school pupils, in which paper and pencil interviewing continues to have a role. Interest in using the web for such surveys may increase.

The challenge of the Internet

The characteristics of government social surveys of the general population do not readily lend themselves to web data collection. Social surveys of the private household population tend to have the following requirements:

- random probability sampling
- long and complex questionnaires, e.g. interviews of 1 hour or more are typical
- high response rates, e.g. 80% plus are a common requirement

These features contrast with common practice on web surveys and raises questions about the potential use of the Internet for general population surveys which are required to yield nationally representative estimates. The absence of an adequate sampling; the fact that the proportion of households with Internet access remains low, and that the on-line population differs in important respects from the general population mean that it is not possible to achieve good population coverage with stand-alone web-surveys.

This suggests that in the immediate future the most likely application of the web in official social surveys is as part of a mixed mode data collection strategy alongside others, such as mail, CAPI or CATI. For general population surveys that have self-completion elements there is growing interest in reporting via the web. Offering a web option for parts of the survey that are currently interviewer-administered raises even more challenging questions, such as related to mode effects and survey response.

SSD has established the Web CASI project to examine the feasibility of using the Internet as a mode of data collection on government social surveys. Work to date has included a review of relevant literature and some initial office testing of Blaise Internet Services, an add-on module of the Blaise survey software package. This paper outlines some of the design and implementation issues faced by developers of web CASI instruments and suggests criteria for the evaluation of survey software.

2. Design and implementation issues

For a survey research organisation a number of the certainties of life in a CAPI and CATI environment come under challenge when we embark upon web CASI. Our users are not trained interviewers familiar with the look and feel of a CAI instrument. Instead, they are members of the public with varying degrees of computer literacy and different expectations, for example, about how to navigate through a form and where to locate key information such as the questions and response categories and any associated instructions or help.

Further, the computing environment in which users will attempt to complete the questionnaire is beyond our control. The environment includes:

- the operating system (e.g. Windows, Macintosh) and its version;
- the web browser (e.g. Internet Explorer, Netscape) and version;
- speed and type of Internet connection.

Thus, a number of design and implementation issues arise in the production of a CASI instrument for use over the Internet. A number of the important ones are discussed below.

Screens and scrolling

There are two main approaches to questionnaire construction that have conventionally been used on web surveys. With a screen-based approach usually one question is displayed on each screen and, each time a respondent answers a particular question, the screen refreshes to display the next question. Scrolling refers to the situation when the whole of the questionnaire is available at one time by allowing the respondent to scroll up and down, using a scroll bar to the right of the screen. The choice of which method to use is often related to the way in which respondents are invited to complete the survey. Screen by screen presentations are often used with on-line surveys where there is the need for continuous contact between the respondent's computer and a web server. Off-line surveys often use a plain HTML (Hypertext Mark-Up Language) form which the respondent is able to download from a website or receive as an email attachment. The pros and cons of on-line and off-line approaches are discussed later. Here we're concerned simply with design issues.

It has been argued that scrolling is more akin to the way in which people use computers and surf the web and for that reason should be preferred (Dillman, 2000). It is true that experienced web users are likely to find a screen by screen presentation more cumbersome. They may also be frustrated if they feel their freedom to navigate is constrained. This may be compounded if the time delay between individual screens refreshing is perceived by respondents to be excessive.

Another advantage of scrolling is that respondents can more easily form a perception of the content and the length of the questionnaire before they begin their task. This makes scrolling more like conventional paper self-completion questionnaires where the respondent can easily flick through the document before answering and can form a judgement about how long it will take them to complete.

For these reasons, Dillman argues that scrolling should be preferred whenever web surveys are being used to replace paper self-completion methods. However, if web surveys are being used in a mixed mode environment, alongside CAPI or CATI, then the case for using a screen by screen construction is stronger. One could argue that it is more akin to the way in which questions are 'served' and answered in these modes.

On-line versus off-line interviewing

An on-line survey is one in which the respondent accesses the questionnaire instrument on a website and completes it, via their web browser, while connected to the Internet. Respondents can be directed to the Universal Resource Locator (URL) either through paper instructions, for example sent by regular mail or left by an interviewer, or in an email, possibly with a hyperlink.

On-line surveys need constant contact between the respondent's computer and the web server upon which the instrument is located. Thus one needs to consider whether or not this requirement is likely to be acceptable to respondents. For Intranet surveys, for example of employees of a business or other organisation, this is likely to be perfectly acceptable. One proviso is that the connection speed between respondent's computer and web server must be sufficient to ensure that there are not unacceptable delays in the processing of individual questions.

However, on-line methods raise a number of issues for general population surveys. The first of these concerns the length of time that a respondent is required to be on-line to complete the survey. This is likely to be of particular concern to those who pay by the minute for their time on-line. One could offer to compensate respondents for the cost of their time on-line but we need more research to find out whether or not this will be perceived as an adequate incentive for all respondents.

As outlined above, on-line surveys can be problematic if the time it takes for the response to one question being transmitted to the server and for the server to respond is deemed to be excessive. For general population surveys, where few households currently have high speed Internet access, this is a key concern. This suggests that on-line surveys are only feasible for general population surveys where there are few questions being asked and the duration of the interview is short.

In an off-line survey, respondents may initially go on-line to download a file, containing the survey instrument, to their local computer, completing the survey off-line before going back on-line to transmit the completed questionnaire back to the host. An alternative is for respondents to be sent a file, either by regular mail on CD or floppy disk or as an attachment to an email. Respondents need to install the file on their computer before answering the questionnaire. Once the questionnaire is completed, respondents can go on-line to return the completed questionnaire. There are other concerns, such as security or computer functionality, that may be relevant to a decision about on-line or off-line surveys. Respondents' competence in the use of the Internet may need to be higher to complete all the required tasks for off-line use than is the case for on-line use.

Off-line surveys are preferred when the survey is longer and more complex. However, off-line surveys can introduce other concerns. For example, the time that it takes for the questionnaire instrument to download to the respondent's computer may be perceived to be excessive. In addition, respondents may

be unwilling to download files to their own hard drive, worried that they may contain viruses or otherwise damage their computer. A possible alternative to downloading is to send respondents a file via regular mail, for example on CD or floppy disk, or as an email attachment. However, as with the download method this still requires some degree of trust in the integrity of the files being sent. It adds to the burden on respondents and for some may be enough to dissuade them from participating. Off-line surveys are likely to work best when the files that need to be installed on a respondent's computer are relatively small. This means that there may be less opportunity to build complex instruments, for example with extensive interactive edits and multi-media if, as a result, respondents are required to download large computer files.

Plain HTML versus advanced programming

The simplest approach to designing web questionnaires is to provide a plain HTML form that respondents can access via their normal browser. However, plain HTML is limited. It is necessary to supplement HTML with the use of advanced programming languages, such as JavaScript, to allow the use of interactive edits and routing and the addition of features such as pop-up help and progress bars.

One could argue that the possibility of including such features represents a radical step forward compared with what is possible with conventional paper self-completion forms. However, the use of advanced programming languages can be problematic since different browsers may be inconsistent in the way in which they interpret Java and JavaScript. Further, the addition of such features necessarily results in the file size of the instrument growing. This impacts on download times and increases the risk of non-response.

It is worth noting that usability tests performed by the US Census Bureau suggested that respondents rarely made use of on-line help. This was true irrespective of how it was presented, for example via buttons, hyperlinks or icons (Kanarek and Sedivi, 1999).

Security

Security is a major issue for government surveys. Respondents to official surveys need to be assured that the data that they give us will be kept secure. At the same time, we need to devise systems that protect the integrity of our own data from hackers (Ramos, Sedivi and Sweet, 1998 and Clayton and Werking, 1998). The US Census Bureau has developed a three level security system including the use of encryption, authentication and a firewall. Encryption provides strong security of data on the Internet while it passes between respondent and the Bureau. Their encryption solution requires respondents to have version 4.0 or higher of either Netscape Communicator or Microsoft's Internet Explorer and enables strong 128-bit encryption. Once received at Census bureau, the data are protected by a firewall that guarantees that others from outside cannot have access to the data (Kanarek and Sedivi, 1999).

There is a tension between the strength of security and survey response. The experience of the US Census Bureau has been that the more stringent the security the lower is the response. In part, this appears to be related to technical issues, such as respondents not having the required browser version or encryption level. It is also possible that requiring the use of usernames and passwords to access a web survey may be problematic for some respondents (Sedivi, Nichols, Kanarek, 2000)

Data quality

One of the attractions of web based data collection, compared with paper self-completions, is the improvement in data quality that is offered with the use of CAI methods. If the technical issues noted above can be overcome, web CASI opens up the possibility for more complex questionnaires to be self-

administered than is possible with paper based approaches. Complex routing can be incorporated in a way that is not feasible in a paper document. Computations and edits can be carried out at the time of the interview and inconsistent or improbable responses checked with the respondent. This is analogous to the move from pencil and paper methods to CAPI and CATI.

An important aspect of data quality is the reduction of measurement error. This arises when inaccurate answers are recorded for respondents. There are different sources of measurement error, such as poor question wording and differences arising from the mode in which the interview is conducted. It is possible that there will be mode effects arising from the switch from other modes to web CASI. For example, compared with CAPI or CATI, there is a difference between the way in which information is presented to respondents – a move from aural to visual presentation. The speed with which respondents read and respond to questions can't easily be controlled. Clearly more research is needed in this area to examine such questions.

Survey response

Currently, web surveys have tended to achieve lower response than mail surveys. The US Census Bureau has found no evidence from its trials, largely among businesses, that offering a web option in addition to other modes leads to an overall increase in response rate (Sedivi, Nichols and Kanarek, 2000). Part of this is likely to be accounted by technical problems discussed earlier. Another possible explanation offered relates to respondents' concerns about privacy and confidentiality (Couper, 2000).

Response rates to self-completion surveys tend to be lower than to interviewer administered surveys and there is every reason to expect that this will be the case for web surveys too. CAPI and CATI achieve higher response than mail surveys because of interviewer involvement in the initial recruitment of respondents and in sustaining respondent involvement. Interviewers can be good at persuading respondents to participate by explaining the value of the survey and the respondents' participation. During the interview itself, interviewers can develop a rapport with respondents that encourages them to continue. For conventional surveys, the proportion of respondents who refuse to complete an interview once started is low. This contrasts with web surveys. Opinion on the optimal length of web surveys is divided but anecdotal evidence suggests that 15-20 minutes is as much as one can expect. This compares with CAPI/CATI surveys that can range from 30-40 minutes up to 90 minutes or more.

One interesting feature about the analysis of the profile of the on-line population is the over-representation of younger adults. This is one sub-group which tends to be under-represented in random probability sample surveys of the general population. Web CASI might be a way to increase response in this sub-group. There is little evidence on this idea, but what there is tends to suggest that there may be limited or no gains, in mixed mode surveys, from offering respondents a choice of response mode. For example, the US Census Bureau tested whether or not response was boosted by offering both mail and telephone options. Overall 5% of the sample responded by phone but the overall level of response (mail and phone combined) was the same as for when only a mail option was offered (Dillman, Clark and West, 1995).

Questionnaire designers need to be aware that the use of more advanced techniques may impact on respondents' ability to respond. The more complex the questionnaire instrument is, and the more that elaborate programming is embedded within it, the greater will be the size of the resulting file that respondents will need to download to run the questionnaire. This will impact on the time it takes to download the questionnaire and is very likely to impact on response.

Design principles

It has been argued that the same visual design principles that apply to the design of paper questionnaires apply for web surveys (Dillman, 2000). Respondents have the same requirement for information that is presented clearly and efficiently. Layout and design should aim for respondents to read every word of each question and in a prescribed order.

Dillman has developed a set of 14 design principles for web surveys. These are:

1. Introduce the web questionnaire with a welcome screen that is motivational, emphasises the ease of responding, and instructs respondents about how to proceed to the next page
2. Provide a PIN number for limiting access only to people in the sample
3. Choose for the first question an item that is likely to be interesting to most respondents, easily answered, and fully visible on the welcome screen of the questionnaire
4. Present each question in a conventional format similar to that normally used on paper self-administered questionnaires
5. Restrain the use of colour so that figure/ground consistency and readability are maintained, navigational flow is unimpeded, and measurement properties of questions are maintained
6. Avoid differences in the visual appearance of questions that result from different screen configurations, operating systems, browsers, partial screen displays, and wrap-around text
7. Provide specific instructions on how to take each necessary computer action for responding to the questionnaire, and give other necessary instructions at the point where they are needed
8. Use drop-down boxes sparingly, consider the mode implications, and identify each with a “click-here” instruction
9. Do not require respondents to provide an answer to each question before being allowed to answer any subsequent ones
10. Provide skip directions in a way that encourages marking of answers and being able to click to the next applicable question
11. Construct web questionnaires so they scroll from question to question unless order effects are a major concern, or when telephone and web survey results are being combined
12. When the number of answer choices exceeds the number that can be displayed in a single column on one screen, consider double-banking with an appropriate grouping device to link them together
13. Use graphical symbols or words that convey a sense of where the respondent is in the completion process, but avoid those that require significant increase in computer resources
14. Exercise restraint in the use of question structures that have known measurement problems on paper questionnaires, such as check-all-that apply and open-ended questions

Another view is that the web is a fundamentally different medium from paper and that much more research is required before we can determine optimal designs (Couper, 2000). Couper suggests that design issues interact with the type of web survey being conducted and the population which is being targeted. For example, an optimal design for a survey of teenagers may be quite different from that which is required for a survey of elderly people.

Dealing with the variety of computer hardware and software that may be used by respondents to access the survey also needs to be kept in mind. A range of factors, such as operating system, browser type and version and Internet connection speed may affect the functionality of the survey instrument. Most of these are outside our control. The result may be that some respondents may not be able to access the questionnaire at all, others may find it slow to respond and/or the questionnaire does not display as intended. One solution to these problems, employed by Statistics Canada in some of the web surveys that they have undertaken, is to build a customised web browser and deliver it to the respondents with the

questionnaire application. The drawback of this approach is that it then requires respondents to download, or install, larger files and this may impact adversely on response.

3. Criteria for evaluating web data collection software

Following on from the above discussion, we can start to think about the sort of criteria that we might use to evaluate the efficacy of web data collection software. The following criteria are provisional and we would welcome comments and views. They have been informed by the specific needs of Social Survey Division (SSD) of the ONS and some may be particular to our context as a government agency. There are possible tensions

a) Provide integration with CAPI/CATI software

Given that it is our expectation that web CASI will initially develop as part of a mixed mode design, we want any product to integrate with software used for CAPI/CATI. Ideally, it should be possible to produce a web CASI questionnaire from the same data model used to produce a CAPI or CATI instrument.

b) Easy to use

The software should be easy to use for questionnaire developers.

c) Sufficient functionality

Any software product should have the same functionality as we would expect from any CAI product, e.g. automated routing, range checking, programmable checks on consistency, computer assisted coding.

d) Form navigation and form design

The product should allow for flexibility in the look and feel of the data entry window. Both a screen by screen presentation and a plain HTML form are required. The developer should have choices about how to allow the respondent to navigate through the form, e.g. option of using previous/next question/page buttons including the option to disable comparable browser buttons. Ideally, there should be options to mix visual presentation of questions and answers, e.g. through use of radio dials, drop down boxes, tick boxes; whether or not and to include progress monitors (e.g. bars, questions on route information). Other useful options include decisions about the provision of interactive help, e.g. with choice of tags, buttons and pop-ups. Question and answer categories should remain in view and wrap round when viewing window is resized by user.

e) On-line and off-line modes

Both on-line and off-line data entry should be supported.

f) Not platform or browser specific

In the developers' and users' environment, the software should not be dependent on the use of a particular platform or browser. Or at the very least, the software should have full functionality in the dominant platform and browser environment.

g) Works well with low speed Internet connection

In the UK, to date there has been low take-up of high speed Internet access amongst domestic consumers. Therefore, web CASI software should have the expectation that most users will be accessing the instrument via a low speed Internet connection.

h) Application is robust

In production environment, it is critical that the application is robust, e.g. that multi-user access doesn't create adverse response times or the web server to crash.

i) Allows remote administration

The ONS is split across five sites in England and Wales and the server which will host a web survey may not be co-located with the developers of the instrument. Therefore, we require that our web CASI software has the facility to enable the installation, configuration and maintenance of the questionnaire instrument, and associated databases, to be carried out remotely, i.e. away from the web server itself.

j) Security enabled

In government surveys, security of respondents' data is a key concern. Software should provide for data encryption and support common security applications, e.g. Secure Socket Layer (SSL).

k) User authentication

There is a need to provide for a system of user authentication, e.g. to ensure correct respondent completes a form once only.

4. An evaluation of Blaise Internet Services (IS)

In Social Survey Division of ONS we have been using CAI in our surveys since the early 1990s. Blaise is our preferred CAI software product and we are keen to use Blaise for CASI via the Internet. Blaise IS has been produced by Statistics Netherlands as an add-on module of Blaise. In this section, with reference to the criteria listed in the previous section, we describe our experience of using Blaise IS. To date, we have restricted our work to office testing of the beta version of Blaise IS 1.1. Office testing does not provide an adequate environment to test all of the criteria but allows us to comment on some important aspects.

Blaise IS overview

Blaise IS produces an internet-ready questionnaire using a standard Blaise data model that has been prepared and compiled in Blaise for Windows **version 4.3 or higher**.

Blaise IS provides two methods of producing a questionnaire:

1. Blaise IS System Manager produces a dynamic HTML form which presents each question screen by screen to respondents, or
2. An HTML Generator produces a plain form in HTML in full view.

The dynamic HTML form has been designed for use with on-line interviewing and it requires that the respondent's computer is in constant contact with the host server. In this version, automated routing, range and consistency checks are supported. The plain HTML form can be completed off-line and can be sent to respondents as an e-mail attachment or downloaded from a website. The form is static and therefore it is not possible to include automated routing or consistency checks.

Blaise IS requires Microsoft Internet Information Server version 4 or higher at the server end.

Evaluation of Blaise IS

For the purposes of testing we developed a questionnaire based on an existing data model used for one of our CAPI surveys.

We were able to produce a web version of the questionnaire using essentially the same data model that had been prepared for our CAPI survey. However, we did not find it easy to use the Blaise IS System Manager/HTML Generator. The absence of full documentation meant that it was necessary to contact Blaise Support before we could successfully install the questionnaire on our pseudo 'web server'. A

particular problem, for us because we can't be sure of having day-to-day access to the web server, is the lack of a remote administration feature. We were able to find a workaround solution by ignoring all the system manager error messages that were being generated.

Version 1.1 of Blaise IS does not have the full functionality that we require. For example, it does not allow the use of external files and it is not possible to use checks/signals with the static web form.

The look and feel of the web forms are, in our view, sub-optimal. Figure 1 shows an example page generated by the Blaise IS System Manager of the dynamic HTML form. With the dynamic HTML form a respondent is presented with a question per page.

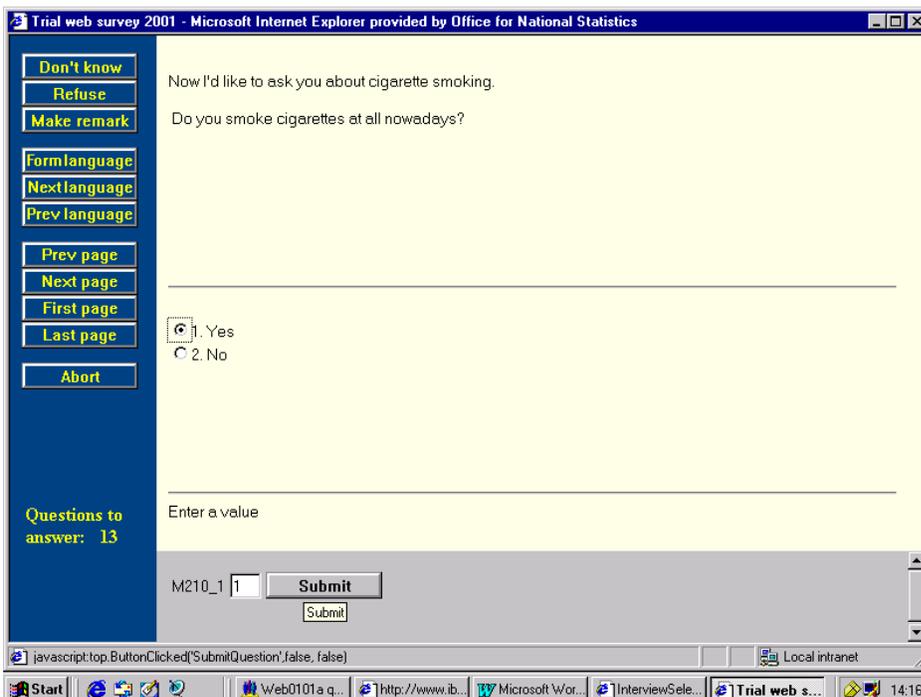
The default screen layout, we feel, is not suitable for respondents. It is too busy and we would like the option to remove, and re-position, the buttons down the left-hand side of the screen.

Using the mouse to point and click is a convenient way for respondents to submit answers. However, if one wishes to use the keyboard, the tab key must be used to navigate around the screen; an approach which is fine for trained interviewers but not necessarily intuitive to respondents. In addition, the <Enter> and <Esc> keys do not perform their normal Windows function which again may be counter-intuitive.

The content and presentation of error messages (Figures 3 and 4) are other areas for design improvement. We found that, on occasions, it is necessary to scroll across the screen to view the whole question because text has not wrapped round (Figure 5). (our office monitors were set at 800x600).

The look of the plain HTML form is less problematic but the positioning of answer boxes was sometimes out of vision (Figure 6).

Figure 1 Question presentation with dynamic HTML form



As our work, to date, has been within the office we have only tested Blaise IS with Internet Explorer version 4 and 5. It is perhaps unrealistic to expect that web survey software will work properly on all platforms and with all versions of all browsers. However, it is most important that the vast majority of potential respondents can use the software.

Blaise IS has been tested by Statistics Netherlands with version 4 of Internet Explorer and Netscape run on a Microsoft Windows platform.

Figure 2 Entry out of range error message

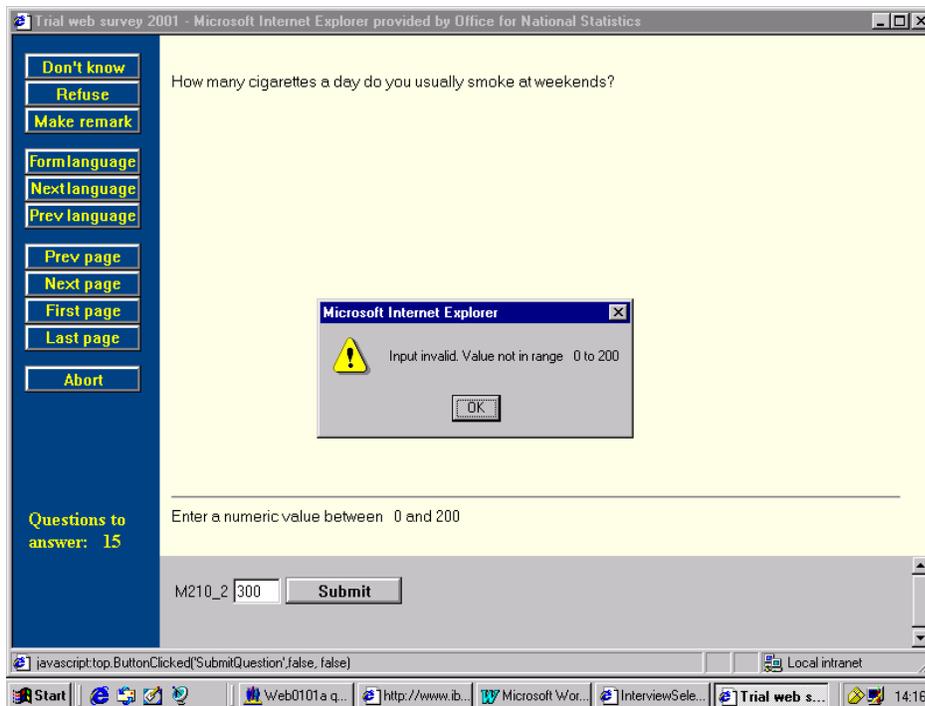


Figure 3 Signal check message

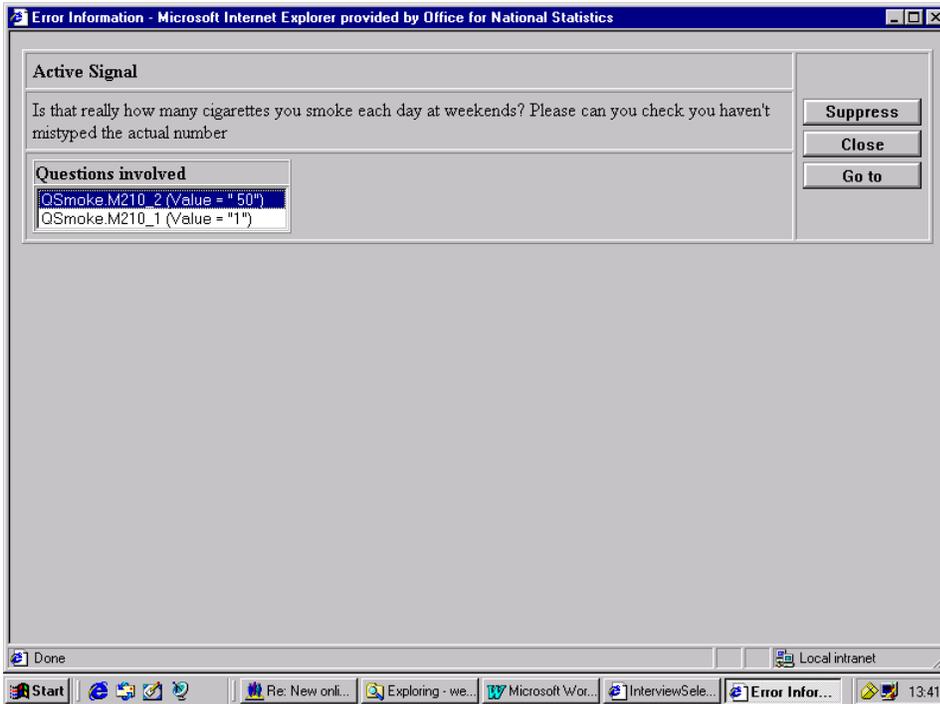


Figure 4 Failure of question text to wrap round

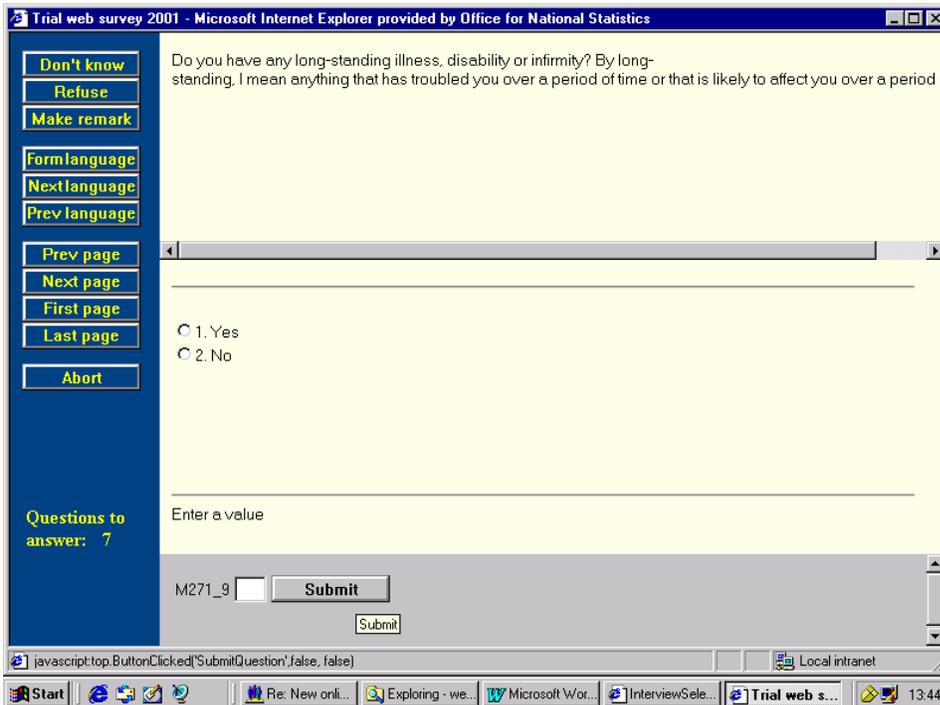


Figure 5 Question presentation with static HTML form

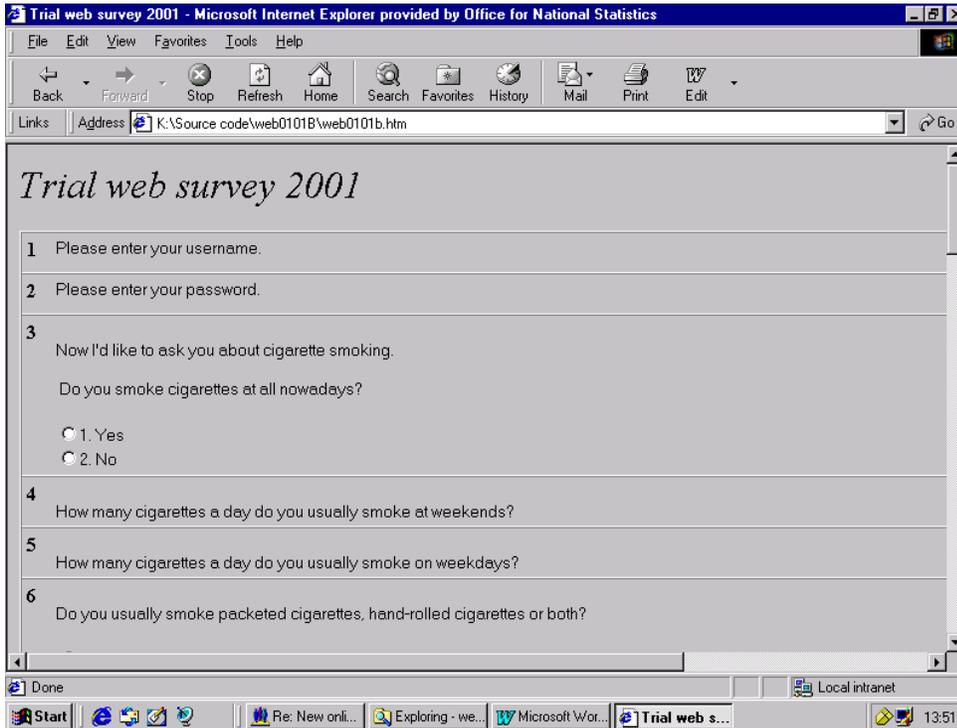
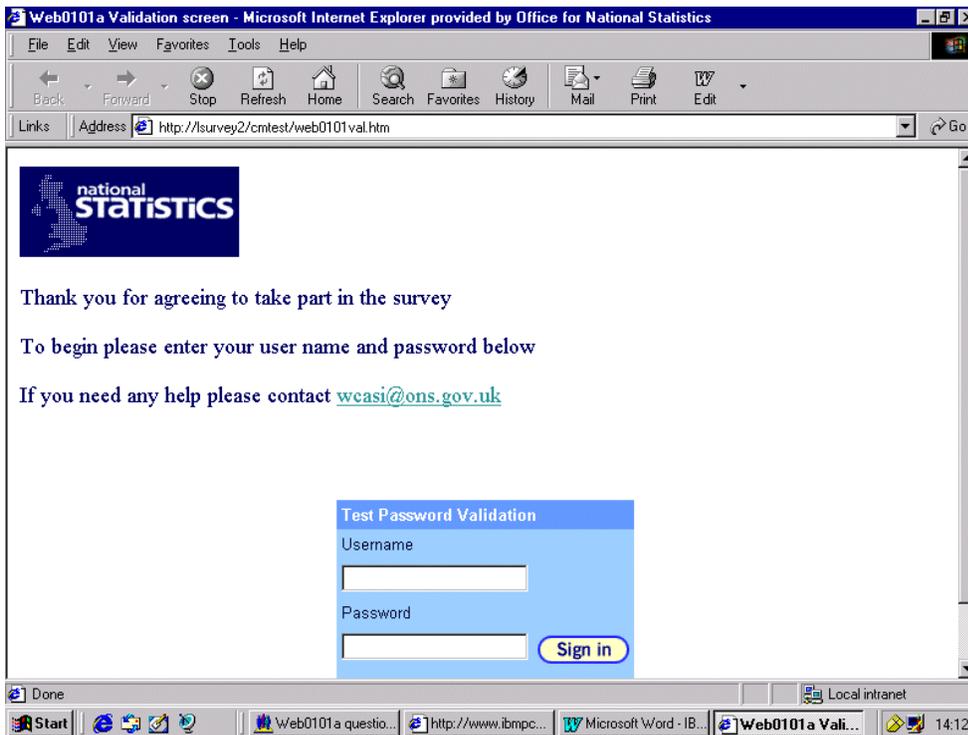


Figure 6 User authentication



Blaise IS supports Secure Socket Layer, a level of data encryption that meets our requirements for the security of data being transmitted.

Blaise IS provides a default Gateway, or welcome, page for respondents. Ahead of a Gateway page, we will need to authenticate users and we are thinking we might handle this based around a system of usernames/passwords. Figure 2 shows a validation page which we have developed. This is an HTML form which runs 3 checks against a survey database:

1. Username exists;
2. Respondent has not completed the survey previously;
3. Correct password for the username.

A successful authentication sends respondents to the Blaise Gateway page. Once into the questionnaire, we need a method to identify who is responding to the survey and we would like to avoid the need for respondents to type in an individual identifier, such as a serial number.

Table 1 summarises our initial evaluation of version 1.1 of Blaise IS. Overall, our evaluation suggests that there is more development needed of the software before we could contemplate using it in a production environment.

Table 1 An initial evaluation of Blaise IS version 1.1

Criteria	Blaise IS version 1.1
a) Provide integration with CAPI/CATI software	√√
b) Easy to use	√
c) Sufficient functionality	X
d) Form navigation and form design	X
e) On-line and off-line modes	√
f) Not platform or browser specific	√
g) Works well with low speed Internet connection	?
h) Application is robust	?
i) Allows remote administration	X
j) Security enabled	√√
k) User authentication	X

Key: √√ = Meets our requirements, √ = Nearly meets our requirements X = Doesn't meet our requirements, ?= Can't comment at this time

5. Conclusion

This paper has suggested that the design and implementation issues that arise when developing a CAI instrument for self-completion over the Internet are considerable. In the case of functionality, such as automated routing, range and consistency checks, the needs are the same as for a CAI instrument administered in other modes, such as CAPI or CATI. In other areas, for example form layout, error messages and interactive help, the self-completion element introduces new demands.

The challenges for CAI software developers are also considerable. This paper has suggested the sort of criteria against which software will be judged and applied these to Blaise IS version 1.1. Our initial evaluation is that Blaise IS requires further development before we can be confident we can use it in a

production environment. We understand that a number of the deficiencies identified are to be addressed in the release of version 2.0 of Blaise IS. We understand the release of this version has been pushed back some months and we anxiously await its release.

References

Couper MP (2000) Web Surveys: A Review of Issues and Approaches Public Opinion Quarterly Volume 64, American Association for Public Opinion Research.

Clayton RL and Werking GS (1998) Business Surveys of the future: the world wide web as a data collection methodology in Couper MP, Baker RP, Bethlehem J, Clark CZF, Martin J, Nicholls II W L and O'Reilly JM (eds) Computer Assisted Survey Information Collection, John Wiley and Sons.

Dillman D, Clark JR and West KK (1995) Influence of an invitation to answer by telephone on response rates to census questionnaires Public Opinion Quarterly, Volume 58, American Association for Public Opinion Research.

Dillman D (2000) Mail and Internet Surveys, John Wiley.

Kanarek H and Sedivi B (1999) Internet Data Collection at the U.S. Census Bureau, Paper to Federal Committee on Statistical Methodology conference.

Ramos M, Sedivi B and Sweet E M (1998) Computerised Self-Administered Questionnaires in Couper MP, Baker RP, Bethlehem J, Clark CZF, Martin J, Nicholls II W L and O'Reilly JM (eds) Computer Assisted Survey Information Collection, John Wiley and Sons.

Sedivi B, Nichols E and Kanarek H (2000) Web-based collection of economic data at the U.S. Census Bureau Paper presented to ICES II Conference.

Acknowledgements

The author would like to acknowledge colleagues in the Survey Computing branch of ONS for their help in formulating the ideas that went into this paper. Specifically, thanks are due to Colin Miceli, Tom Bryson and Steve Edwards.

About the Author

John Flatley is a Senior Researcher in the Social Survey Division of the Office for National Statistics. He is Manager of SSD's Web CASI project which is examining the feasibility of utilising the Internet as a mode of data collection in government social surveys. For more information contact john.flatley@ons.gov.uk, telephone: from UK: 0207 533 5530; international +44 20 7 533 5530

Session 7: Blaise API

- Blaise API, A Practical Application
Andrew Tollington and Pauline Davis
Office for National Statistics, United Kingdom
- Some Examples Using Blaise Component Pack
Gina Qian Cheung, University of Michigan, USA
- Internet Interviewing Using Blaise API
Bas Weerman, CentERdata, University of Tilburg, The Netherlands

Blaise API, a practical application.

Pauline Davis, ONS
Andrew Tollington, ONS

Keywords

Application Programming Interface (API)
Component Object Model (COM)
Blaise [meta] data
Windows

Introduction

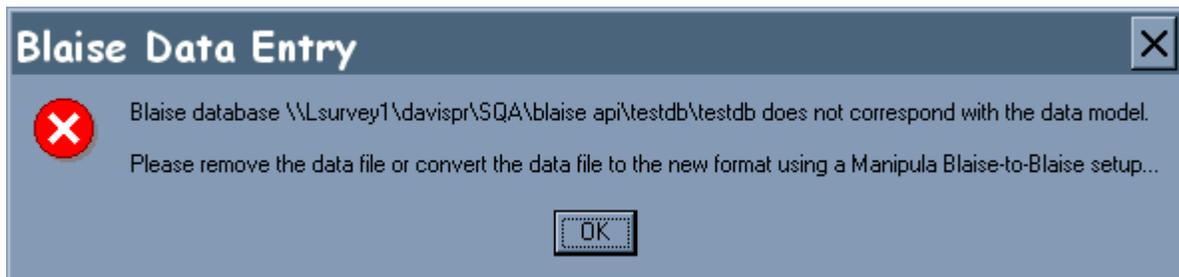
The arrival of an API for Blaise will enable developers to design and build powerful Windows applications that not only harness the core Blaise engine but extend and achieve functionality that would otherwise be complex - if not impossible, time-consuming and costly - using traditional Manipula/Cameleon programming techniques.

The Blaise API provides total access to your data and meta-data in a structured, object based manner. In exposing a COM interface it can be implemented with any supporting language of your choice from JavaScript to COBOL.

The scope of the practical application of this technology is seemingly limitless and this paper will discuss the design and build of an example application that deals with a typical problem and demonstrates some of these points. It is not intended to delve deeply into COM programming or every facet of the Blaise Object Model but to provide an introduction to the many possibilities now available for the Windows Developer and their value to the Blaise programmer.

The example application - data file incompatibility problems

Blaise checks to see whether the current data description is compatible with the current data file. If it isn't, when you try to run the Data Entry Program (DEP) you get an incompatibility error message similar to the one given below.



Incompatibility between the data model and the data file can occur for a number of reasons, some of which are when you:

- Change the number of CHECKS or SIGNALS
- Change the attributes of a field
- Change the number of fields
- Change the order of fields in the FIELDS section
- Change the valid range of fields
- Change the number of choices in an enumerated type
- Change the primary key definition
- Change the secondary field definition

Incompatibility can occur even when making small changes. Whilst developing a questionnaire data file incompatibility is inevitable. If the data file becomes incompatible whilst you are developing it is easy simply to delete the database and start again, but if the questionnaire is already in the field this can prove much more difficult. This is usually solved by sending out a batch file/Visual Basic Script(VBS) with the new version of the questionnaire, that updates the dataset. The majority of the time you can easily create Manipula script to convert the data from one form to another, but if for example a field type has changed this means producing long script which converts each block and then in the block that has changed, each field.

Summary of data file incompatibility problems

1. Sending out incompatible datamodels, sometimes unknowingly and then having to create a quick fix.
2. Having to write long Manipula script to convert from one model to another e.g where field types have changed.
3. Not being able to quickly find out the differences between two data models if you do not have the original code.

Current solutions

1. Sending out a hook with the new version of the questionnaire which runs a Manipula script to convert the data from one version of the questionnaire to another (fig 1.)
2. Writing long Manipula script to convert the current data file to the current data description. (fig 2.)
2. Using the Cameleon, Manipula and Maniplus tools to identify the differences between two datamodels.

All of these solutions are time consuming and prone to error. A better solution would be one that combines access to metadata and data at the same time, that could identify an incompatible datamodel with a questionnaire of the same name already in the field, let you know the differences between two datamodels and where appropriate write out the Manipula script. This is all possible using the Blaise API.

```

TextPad - [P:\SURVEY\GHS0010B\object\objfiles\hook.vbs]
File Edit Search View Tools Macros Configure Window Help
1
2 ' Hook.Vbs
3
4 ' Extension script to select latest datamodel
5
6 ' Usage: WScript //I Hook.Vbs
7
8 ' History:
9 ' 23-Mar-2000 sa -- Initial version WSH Script
10 ' 28-Mar-2000 sa -- Adapted for FMA
11 ' 27-Sep-2000 anf -- Adapted for general purpose use
12
13
14
15 Const Datafolder = "C:\Casebook\Data\"
16 Const Data_Ext = ".Bd"
17 Const Meta_Ext = ".Mi"
18
19 ' ***CHANGE THIS LINE TO DEBUG...
20
21 Const DEBUG_HOOK = false
22
23 ' ***CHANGE THIS LINE TO DEBUG...
24
25 If DEBUG_HOOK Then
26 WScript.Echo "Objmenu/HookScript: Warning! DEBUG_HOOK is True"
27 End If
28
29 Public WShell ' As Object
30 Set WShell = WScript.CreateObject("WScript.Shell")
31 If WShell Is Nothing Then
32 WScript.Echo "Objmenu/HookScript: fatal error (unable to create WScript.Shell)"
33 WScript.Quit 100
34 End If
35
36 Public oFS ' As Scripting.FileSystemObject
37 Set oFS = WScript.CreateObject("Scripting.FileSystemObject")
38 If oFS Is Nothing Then
39 vbOkOnly + vbCritical
40 WShell.Popup "Unable to create Scripting.FileSystemObject", , "Objmenu/HookScript", 0 + 16
41 WScript.Quit 101
42 End If
43
44 Call Main

```

Fig 1

```

Blaise 4.5
File Edit Project Run Database Tools Add-Ins Window Help
\\Lsurvey\davispr\SQA\blaise api\old2new\ver1tover2.man
MANIPULATE
Outfile.QID:=Infile.QID
Outfile.QDatabag:=Infile.QDatabag
Outfile.Qtmadtot:=Infile.Qtmadtot
Outfile.QStart:=Infile.QStart
Outfile.QInter.QHealth:=Infile.QInter.QHealth
Outfile.QInter.QEntdate:=Infile.QInter.QEntdate
Outfile.QInter.QWhback:=Infile.QInter.QWhback
Outfile.QInter.QLjbent:=Infile.QInter.QLjbent
Outfile.QInter.QPaint:=Infile.QInter.QPaint
Outfile.QInter.QMDExp:=Infile.QInter.QMDExp
Outfile.QInter.QWhforw:=Infile.QInter.QWhforw
Outfile.QInter.QLJBwl:=Infile.QInter.QLJBwl
Outfile.QInter.QSearch:=Infile.QInter.QSearch
Outfile.QInter.QBarrier:=Infile.QInter.QBarrier
Outfile.QInter.QSComp:=Infile.QInter.QSComp
Outfile.QInter.QChildCr.Introchr:=Infile.QInter.QChildcr.Introchr
Outfile.QInter.QChildCr.CCpast:=Infile.QInter.QChildcr.CCpast
Outfile.QInter.QChildCr.CCschr:=Infile.QInter.QChildcr.CCschr
for i:= 1 to 16 do
    Outfile.QInter.QChildCr.Cicare[i]:=Infile.QInter.QChildcr.ccarel[i]
enddo
Outfile.QInter.QChildCr.ccarp1:=Infile.QInter.QChildcr.ccarp1
Outfile.QInter.QChildCr.ccaral:=Infile.QInter.QChildcr.ccaral
Outfile.QInter.QChildCr.cbrak1:=Infile.QInter.QChildcr.cbrak1
for i:= 1 to 16 do
    Outfile.QInter.QChildCr.c2care[i]:=Infile.QInter.QChildcr.ccare2[i]
enddo
Outfile.QInter.QChildCr.ccarp2:=Infile.QInter.QChildcr.ccarp2
Outfile.QInter.QChildCr.ccar2:=Infile.QInter.QChildcr.ccar2
Outfile.QInter.QChildCr.cbrak2:=Infile.QInter.QChildcr.cbrak2
for i:= 1 to 17 do

```

Fig 2.

COM very briefly

COM stands for Component Object Model, or more precisely Microsoft® OLE Component Object Model. This technology has been part of the foundations of Windows application development for eight or so years. With the forthcoming .NET (dot-net) Framework, COM is being replaced by new and very different technology. Microsoft® are committed, however, to supporting COM for the foreseeable future such is the industry's reliance on what has been, and will remain, a fundamental technology in Windows development.

COM can be thought of as a 'Black Box' of functionality. A compiled program whose functionality can be accessed by other programs could be a COM Component. The internal implementation of the Component is of no concern. In order to build the driver's cockpit of a car the engineer does not need to know how to build an engine or how the front wheels steer, all she needs is to know how to connect the steering wheel and throttle to the relevant areas of the machine. COM displays a similar distinction between functionality that is hidden from the user - and Interface - exposed to the user. The Blaise API is presented to users via a COM interface. Any COM aware language can, therefore, bind to the component and make use of its functionality (see appendix A)

How might the Blaise API provide a better solution to data compatibility problems

The usual way in which a Blaise Datamodel is tested for compatibility with a database or another datamodel is basically to try it. If it works...then, well it works. If it doesn't then it's not compatible. Often this is a manual test: one which is neglected from time to time and has on occasion caused very time-costly problems. This is not ideal. The Blaise API Component provides us with the possibility of a much better solution, one which can be automated if need be and run behind the scenes without the involvement of Manipula or Cameleon.

What a COM solution looks like

One solution is to build another component. If we can wrap up the relevant functionality of the Blaise API together with the logic that suits our business environment then we have our solution. So at the core is BLAPI4A.DLL (the Blaise API). Sitting on top is our component developed in MS Visual Basic® 6.0, which currently has many different names - lets call it BLWRAP.DLL. On top of that is any client wishing to make use of it (remember we are still in COM-land so 'any client' really can be 'any'). Other major considerations are that the component is lightweight, simple and very easy to use.

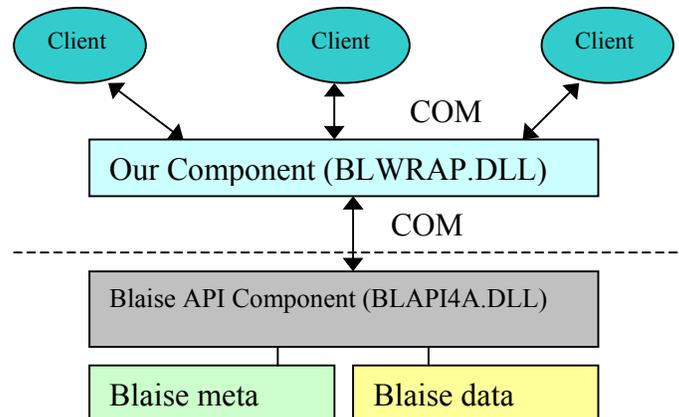


Fig 3.

Figure 3. shows the simple relationship between Blaise and our client application(s).

So the core of our solution has been identified as BLWRAP.DLL. This is what client applications will have to bind to if they wish to make use of the proposed functionality above. How is this functionality to be implemented in our component?

The COM Interface for our component

Simple, lightweight and very easy to use: these are all major goals in the design of our component. If it were over complicated then there would be no point using it - you may as well go straight to the Blaise API. For the purpose of this paper we will illustrate only the functionality concerning compatibility issues. As discussed earlier we need to have a mechanism through which two datamodels and data can be assessed for compatibility. In a simple usage the client may only need to know 'are they compatible?' A more demanding client may need to know 'Why not?' if the answer is 'No'.

A suitable interface definition could be a method as below:

```
Compare (data1,data2) As Diffs
```

data1 and data2 are the datamodels to be compared, Diffs is a return collection of Difference objects. The Difference class describes a difference between data1 and data2 that results in incompatibility. This method can be used as follows;

```
'Constants to point to the two datamodels we are interested in.
```

```
Private Const DATA_MODEL1 As String= "C:\\WORKFILE\\DATA\\ONE0009A"  
Private Const DATA_MODEL2 As String= "C:\\WORKFILE\\DATA\\ONE0009B"
```

```
Sub Main()
```

```
Dim cmpData As New BlaiseCompatibilty  
Dim cmpDiff As Difference  
Dim cmpDiffs As Diffs
```

```
Set cmpDiffs = cmpData.Compare (DATA_MODEL1,DATA_MODEL2)
```

```
If cmpDiffs.Count = 0 Then
```

```
    ' No differences - datamodels are compatible
```

```
Else
```

```
    ' There are differences - datamodels are  
    ' incompatible
```

```
For Each cmpDiff In cmpDiffs
```

```
    ' Iterate through the differences and process  
    ' the data as required.
```

```
Next
```

```
End If
```

```
End Sub
```

Fig 4.

The ability to write small routines similar to Figure 4. is a great advantage. Almost every system we run will benefit from being able to dynamically assess compatibility between two datamodels - especially when they are seemingly the same version and previously would not have been checked. No in-depth benchmarking has currently been undertaken but on one of our medium length instruments a similar routine to the one above completes in a couple of seconds, quick enough not to be an issue.

Functionality of the component

How does our component use the Blaise API ? The reliability of our Compare method really depends upon one thing - what makes a datamodel incompatible with another? This paper started by highlighting some of the changes you can make to a datamodel which will render it incompatible with its data; there are doubtless more. This is an obvious area for further development. Once all the rules can be encapsulated within our component, it should be totally reliable.

Difference Objects are returned when the rules for compatibility are broken. The Difference Class discussed above is the member that describes the data specific to an incompatible difference between two datamodels. Internally, however, a Difference Object is created for every detail of a datamodel that may distinguish it from another.

Below is an overview of what happens when 'Compare' is called;

Using the Blaise API Component:

1. Open a database with DATA_MODEL1
2. Create Difference Objects for datamodel level attributes according to the rules.
3. Iterate through every field creating a Difference Object for each, storing its attributes according to the rules.
4. Close the database
5. Repeat 1-4 for DATA_MODEL2
6. Compare the two collections of Differences.
7. The method returns a collection of those which are not identical.

This is the basic mechanism behind the method. It does, however, perform collaborations with other areas of both our component and the Blaise API. It will, for instance, attempt conversion if required when incompatibilities are identified.

This section on functionality may seem to hold no substance - it really is that simple. Previously our Compare method could have been accomplished with various Cameleon and Manipula scripts. Without the messy parsing of the outputs, having the Compare method executing as part of another process and without user-input was impossible.

Summary

As this paper has shown the Blaise API Component offers us the opportunity to achieve real integration of Blaise functionality into our production systems. Our component is just one way to harness the core Blaise engine, it works for us and will no doubt continue to grow in functionality to provide an increasingly seamless integration of Blaise into our computing environment.

What's Next ?

Our agenda includes evaluating functionality for Active Server Pages, building a drag and drop datamodel generator for data conversion and a Blaise Custom Task Component for Microsoft® Data Transformation Services. There's a lot to do, and none-of it with Manipula or Cameleon.

Appendix A

Interoperability between languages is, chiefly, what COM is for. As previously mentioned; Microsoft® will not abandon the technology for a very long time. To highlight this fact the following is a console application which returns the number of records in a Blaise database. This simple application is written in C#. (Pronounced 'C Sharp' a core, brand new development language of Microsoft's impending .NET Framework)

```
using System;
using BlAPI4A;
namespace BlaiseApps
{
    class DataAttributes
    {
        static int Main(string[] args)
        {
            if (args.Length==0)
            {
                Console.WriteLine("Please enter a valid path and for a
                                   Blaise Database.");
                return 1;
            }

            DataAttributes MyClass = new DataAttributes();

            string error;

            double RetVal = MyClass.Records(args[0],out error);

            if (error.Length ==0)
            {
                Console.WriteLine ("{0} has {1} records.",args[0] ,RetVal);
                return 0;
            }
            else
                Console.WriteLine ("Blaise ERROR :" + error);
                return -1;
        }

        public virtual double Records(string FileName, out string error)
        {
            DatabaseManager MyManager = new DatabaseManager ();
            try
            {
                IBlaiseDatabase MyDataBase = MyManager.OpenDatabase(@FileName);
                MyDataBase.Connected = true;

                error = "";
                return MyDataBase.RecordCount;
            }
            catch(System.Runtime.InteropServices.COMException e)
            {
                error = e.Message ;
                return -1 ;
            }
        }
    }
}
```


Some Examples Using Blaise Component Pack

Gina-Qian Y. Cheung
Survey Research Center
University of Michigan

Introduction:

Blaise Component Pack, commonly called BCP, is a COM library. Blaise uses a proprietary database format that is accessible by the Blaise System only. We can use this library to directly access the Blaise database without the need for any intermediate interface or application. As with any other component this provides properties and methods accessible to programmers.

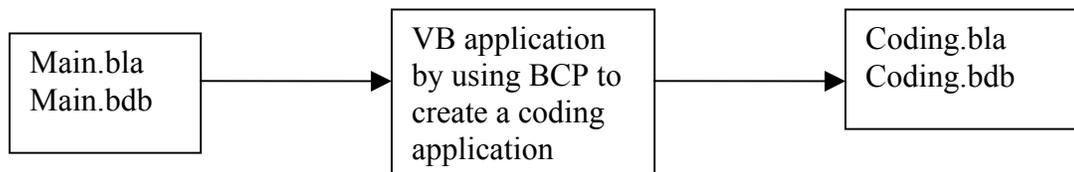
Using this BCP we can do many things with Blaise Database. Some of the main tasks which can be accomplished using BCP are:

- Fetch all the field information (like blockname, field name, field type, display text, primary key, etc.) by reading either Blaise Database(*.bdb) or Blaise Meta Information file (*.bmi).
- Fetch all the data from the database.
- Navigate through all the FIELDS and RULES (using Rules Navigator) of a data model.
- There is no need to write manipula or cameleon scripts. With BCP you can perform a variety of programming and scripting in environments like Borland® Delphi™, Microsoft® Visual Basic®, Microsoft® Visual Basic for Applications (VBA), Microsoft® Visual C++®, Microsoft® Visual J++®, and Sybase®PowerBuilder.

Some examples we had:

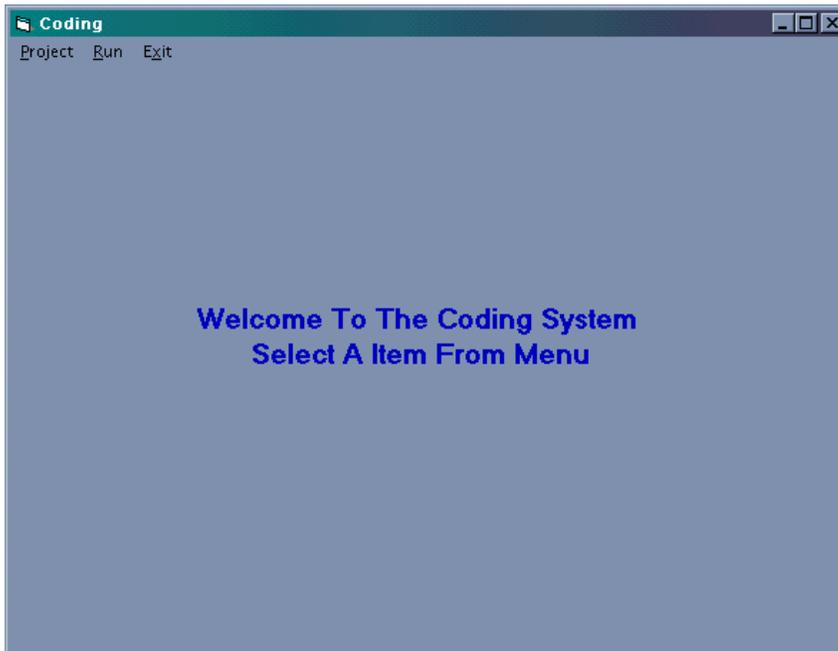
Example 1 : Creating a Blaise Coding Application

Created an application in Visual Basic that creates a Blaise coding program from the main Blaise application. In this application, it lists all the OPEN/STRING fields to the supervisor so that they can select all the fields needed to be coded. Then the Coding Application is created and related data fields are populated from main data set to the coding application data set.

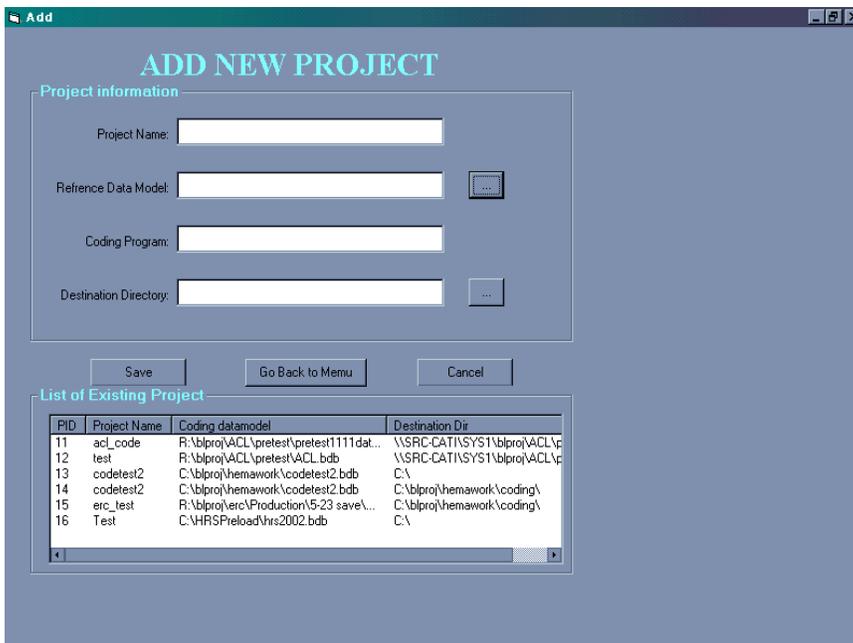


This application creates a coding program after reading a database/data model.

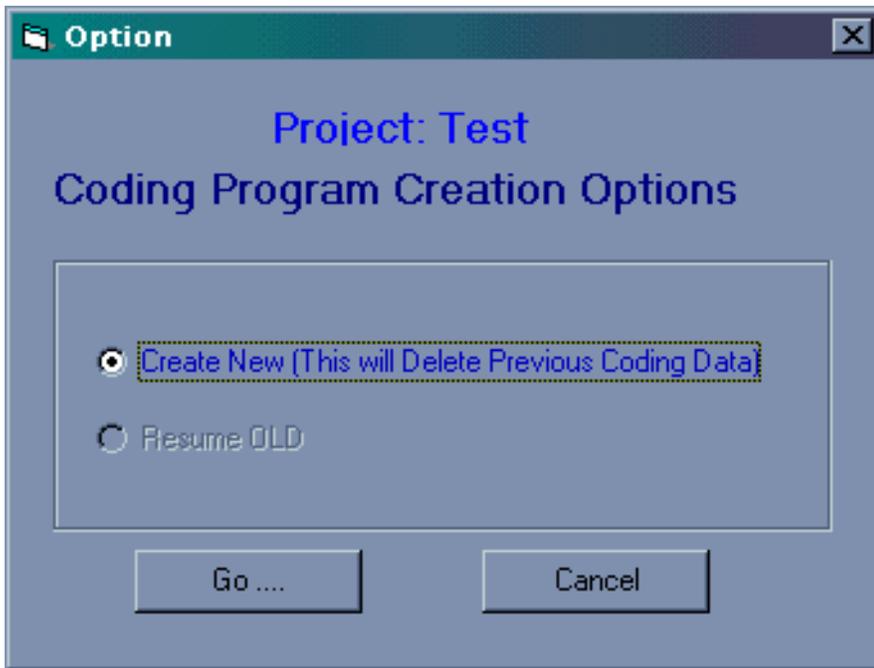
Run the application



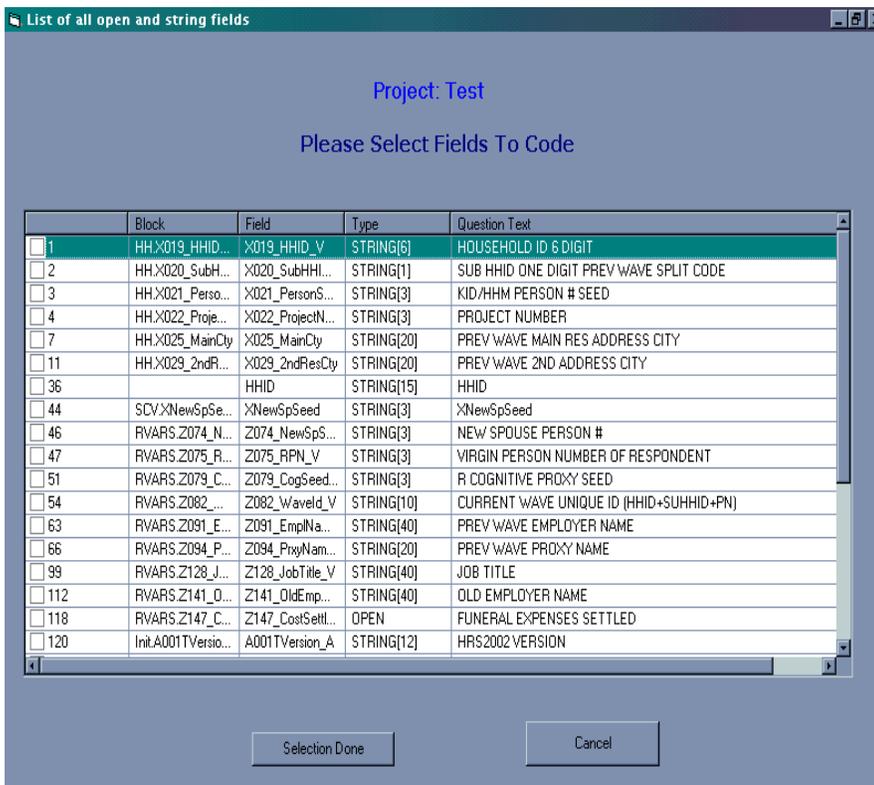
Create a new project (Project -> Add)



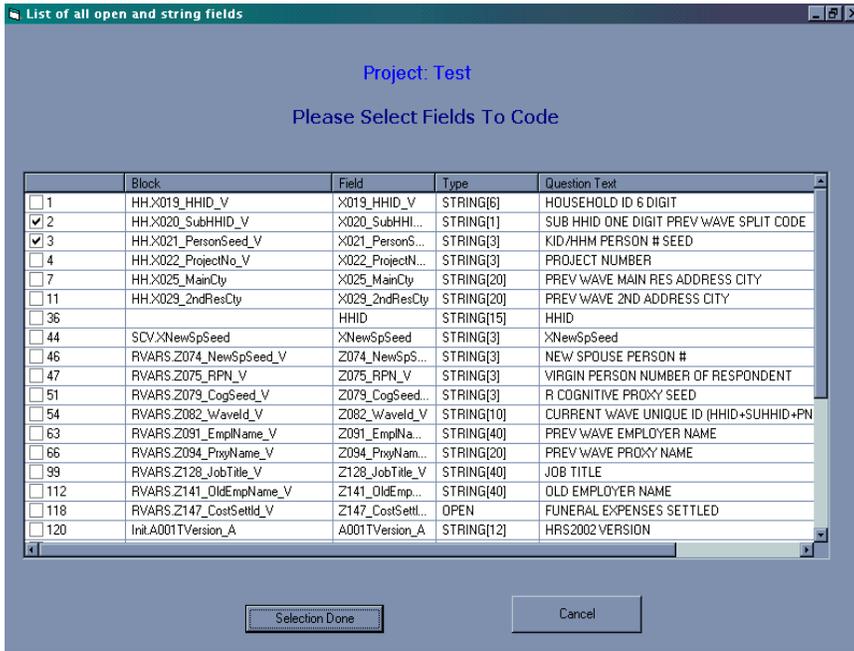
After this click on Project -> OPEN from the main menu



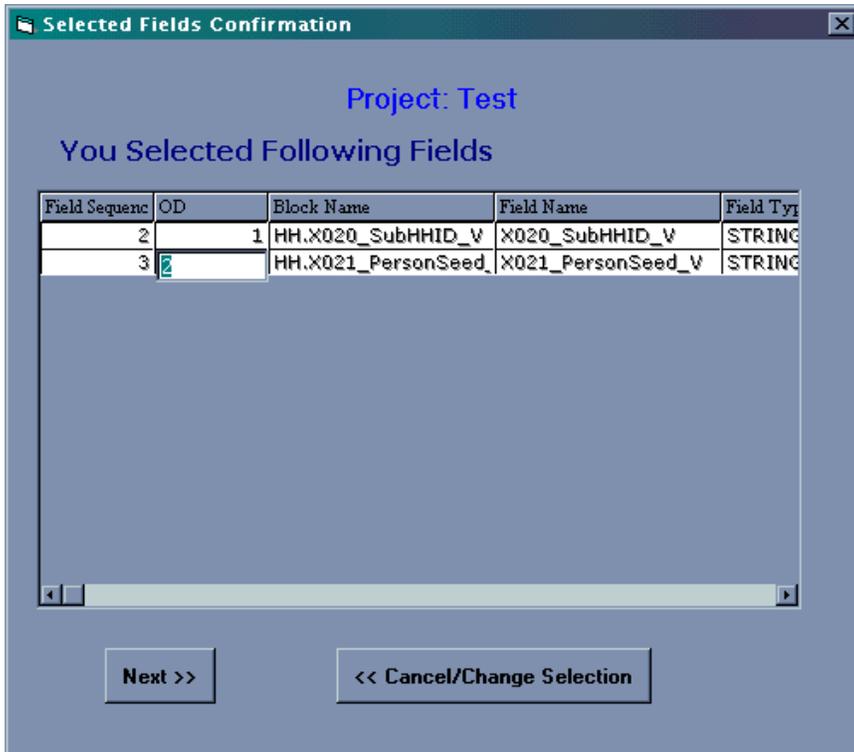
Click on “Go” button
 This displays the list of all open & string fields to code



Select the fields to be coded :



After selecting the fields to Code, click on “Selection Done” button



For each & every field selected, the following screen will come up where you can specify the number of mentions for each field and select the fields to display while coding.

Field Detail

Project: Test

CODING FIELDS DETAILS

Field To Code:

Field Type:

Number Of Mentions:

Fields To Display:

NewFieldName	NewFieldRange	DisplayText	Empty	NoEmp	NoRefu	NoDonr	Refus
X020_SubHHID_V1	0.9	SUB HHID ONE DIGIT PREV WAVE SPLIT CODE	0	0	0	0	0

Change Type:

Field Detail

Project: Test

CODING FIELDS DETAILS

Field To Code:

Field Type:

Number Of Mentions:

Fields To Display:

NewFieldName	NewFieldRange	DisplayText	Empty	NoEmp	NoRefu	NoDonr	Refus
X021_PersonSeed_V1	0.99	KID/HHM PERSON # SEED	0	0	0	0	0
X021_PersonSeed_V2	0.99	KID/HHM PERSON # SEED	0	0	0	0	0
X021_PersonSeed_V3	0.99	KID/HHM PERSON # SEED	0	0	0	0	0

Change Type:

Example 2: View and edit Blaise data

BCP Demo

BCP DEMO

Choose Reference Data Model: ...

Please select the fields from the following list that you want to retrieve based on Key Field

	Block	Field	Type	Question Text
<input checked="" type="checkbox"/>		CurrentUser	String	CurrentUser
<input checked="" type="checkbox"/>		DoneFlag	Enumerated	DoneFlag
<input type="checkbox"/>		IDENT - Primar...	String	IDENT
<input type="checkbox"/>		CaseID	String	CaseID
<input type="checkbox"/>		GFIDNUM	Integer	Coversheet Number
<input type="checkbox"/>		GF090108	Integer	Questionnaire Number
<input type="checkbox"/>		MGID	Integer	Facesheet Number
<input type="checkbox"/>		IDNUM	Integer	First 7 digits of Questionnaire number
<input type="checkbox"/>		MGSTATUS	Integer	Last digit (8th) of Questionnaire number
<input type="checkbox"/>	Attitudes.GF0911	GF0911	Enumerated	GF0911
<input type="checkbox"/>	Attitudes.GF0912	GF0912	Enumerated	GF0912
<input type="checkbox"/>	Attitudes.GF0913	GF0913	Enumerated	GF0913
<input type="checkbox"/>	Attitudes.GF0914	GF0914	Enumerated	GF0914

IDENT

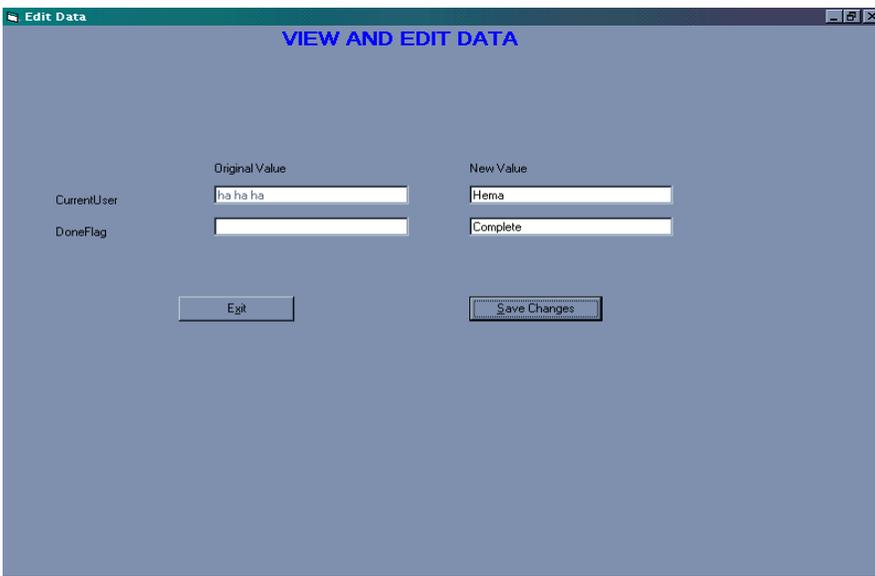
After doing all this click on Get data.

Edit Data

VIEW AND EDIT DATA

	Original Value	New Value
CurrentUser	<input type="text" value="ha ha ha"/>	<input type="text" value="ha ha ha"/>
DoneFlag	<input type="text"/>	<input type="text"/>

Here you can view the existing data from these fields..
 Click on "Save Changes," this will update the data in database if the data is valid as per rules written in Blaise program.



Viewing Data again to see the result :

	Original Value	New Value
CurrentUser	Hema	Hema
DoneFlag	1	1

Buttons: Exit, Save Changes

It's showing 1 in Done Flag as it is enumerated data type. "Complete" is assigned as "1" in UDT(User Defined Data Type). We can display the category text also if it is required.

EXAMPLE 3. BCP Usage and Data/Metadata-out

Purpose: The conditions table, along with the identification of the condition in the variables table, is the identification of the total set of conditions under which the field data is updated in the entire data model. This kind of information is useful in describing or analyzing a questionnaire without the use of a specific programming language such as Blaise. By examining the conditions under which the field is updated, it is possible to verify that the intentions of the designers are being correctly implemented. In addition, by searching the condition table, it is also possible to isolate which fields are dependant on another field being answered. This would be particularly useful when analyzing frequency tables on a dataset. By examining the conditions for a field, you can identify primary info that influence whether or not that question is asked.

The program we wrote, Universe.exe, was written in Visual Basic 6.0 and used the COM object library to analyze a prepared datamodel. We made calls to various BCP functions and methods to gather the information we needed. The program made use of recursive-style calls very much in the manner that Cameleon does. In particular, we used the Rules Navigator functions to traverse the Rules in the datamodel and get information about the Current Conditions (another Rules Navigator function) for each field.

These were the Rules Navigator functions used:

```
MoveToFirstStatement
MoveToThenStatement
MoveToChildStatement
MoveToNextStatement
MoveToParentStatement
MoveToElseStatement
```

In the process of completing our project to produce tables to output Blaise data to SAS in the form of relational tables we used the BCP to produce our set of conditions under which fields in the datamodel were assigned or asked.

These are some of the functions used:

```
' Determine the Rules statement type (IF, ELSE, Assignment, ASK, FOR loop, Block Reference, etc.)
```

```
StatementType
```

```
' Determine the set of Fields involved in the condition
```

```
InvolvedFields.Item(1)
```

```
' Determine the number of nested conditions which apply to the field method
```

```
CurrentConditions.Count
```

```
' Set the condition based on the above count used as an index
```

```
CurrentConditions.Item(nCount)
```

Several other functions were used from the BCP to determine certain field attributes such as

the field type and the field method (ASK, KEEP, SHOW, or assignment).

```
' Get an empty database reference
Set xmDatabaseManager = New BLAPI4A.DatabaseManager
Set xmDatabase = xmDatabaseManager.OpenDatabase("")

' Set the datamodel file name
xmDatabase.DictionaryFileName = xsMetaDataFile

' Determine Field, Auxfield or Local
nFieldKind = xmDatabase.Field(sFieldName).FieldKind

' Determine the base fieldname such as Name instead of HHL.Person.Name
sFieldName = xmDatabase.Field(sFieldName).LocalName
```

Here is the section of code we used for extracting a set of conditions once we have opened the database (datamodel) and navigated to a particular field for which we want the conditions:

```
' Concatenate conditions
sCondition = ""
nConditionCount = xmDatabase.RulesNavigator.CurrentConditions.Count
If nConditionCount > 0 Then
    nCount = 0
    nTerm = 0
    For nLevel = 1 To nStmtLevel
        If nStmtType(nLevel) = blstCondition Or _
            nStmtType(nLevel) = blstElse Or _
            nStmtType(nLevel) = blstForLoop Then
            nCount = nCount + 1
            If nCount > nConditionCount Then
                MsgBox ("Count exceeds ConditionCount")
            End If
        End If
        If nStmtType(nLevel) = blstCondition Or _
            nStmtType(nLevel) = blstElse Then
            sConditionItem = xmDatabase.RulesNavigator.CurrentConditions.Item(nCount)
            If nStmtType(nLevel) = blstElse Then
                sConditionItem = "NOT(" & sConditionItem & ")"
            End If
            nTerm = nTerm + 1
            If nTerm = 1 Then
                sCondition = sConditionItem
            Else
                If nTerm = 2 Then
                    sCondition = "(" & sCondition
                End If
                sCondition = sCondition & ") AND (" & _
                    sConditionItem
            End If
        End If
    Next
End If
```

```

    If nTerm > 1 Then
        sCondition = sCondition & ")"
    End If
End If

```

This is a section of a listing of the output that we get from this program as a part of the description of what the variables in the datamodel look like by field name:

DATAMODEL Demo1

Type

```
TYesNo = (Yes (1), No (5))
```

Locals {These will not be output in conditions}

```
I : INTEGER
```

AuxFields {These will not be output in conditions}

```
Introduction "This is the introductory text"
```

```
: STRING[1]
```

BLOCK block1

```
FIELDS {For Block 1}
```

```
Driver "Do you drive a car or truck?"
```

```
: TYesNo
```

```
CarTruck "Car or truck?"
```

```
: (Car (1), Truck (2))
```

RULES

```
Driver
```

```
If Driver = Yes Then
```

```
CarTruck
```

```
Endif
```

ENDBLOCK

```
FIELDS {For Main Block}
```

```
RWilling "Is the respondant willing to take the interview?"
```

```
: TYesNo
```

```
Field1 : Block1 {A Block of Fields}
```

```
FavColor "What is your favorite color?"
```

```
: STRING[20]
```

```
HavePet "Do you have a pet?"
```

```
: TYesNo
```

```
PetName "What is your Pet's Name?"
```

```
: STRING[20]
```

RULES

```
IF Sysdate > Todate(2000,01,01) THEN
```

```
Introduction
```

```
RWilling
```

```
IF RWilling = Yes OR RWilling = DK THEN
```

```
Field1
```

```
FavColor
```

```
HavePet
```

```
Endif
```

ENDIF

IF HavePet = Yes THEN

 PetName

ENDIF

ENDMODEL

Blaise Data Entry - C:\blproj\hrs2002\Demo1

Forms Answer Navigate Options Help

Is the respondent willing to take the interview?

1. Yes
 5. No

Introduction 1
RWilling 1

Driver
CarTruck
FavColor
HavePet
PetName

New 1/1 Modified Dirty Insert Demo1

Blaise Data Entry - C:\blproj\hrs2002\Demo1

Forms Answer Navigate Options Help

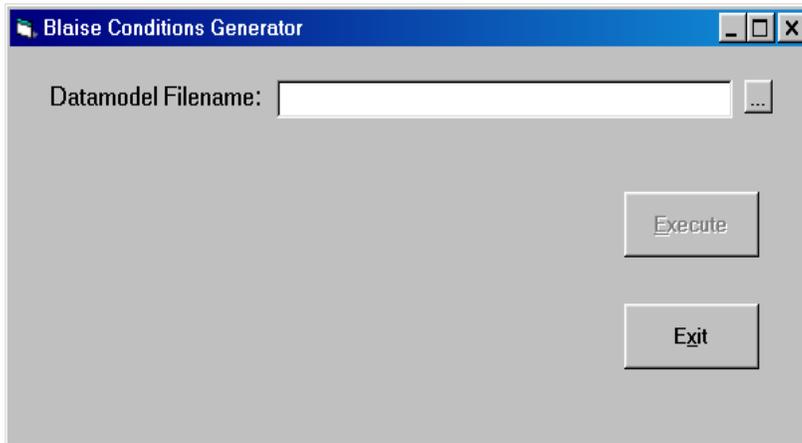
What is your Pet's Name?

Enter a text of at most 20 characters

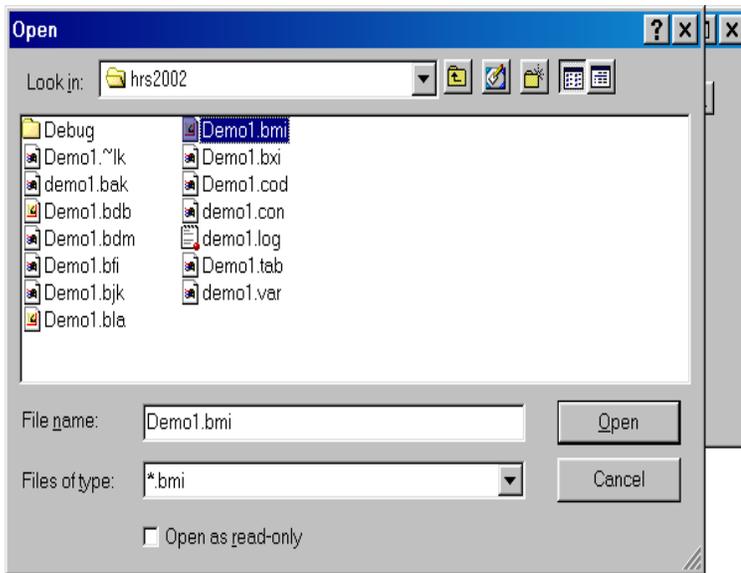
Introduction 1
RWilling 1 Yes
Driver 1 Yes
CarTruck 1 Car
FavColor Red
HavePet 1 Yes
PetName Lassie

New 1/1 Modified Dirty Insert Demo1

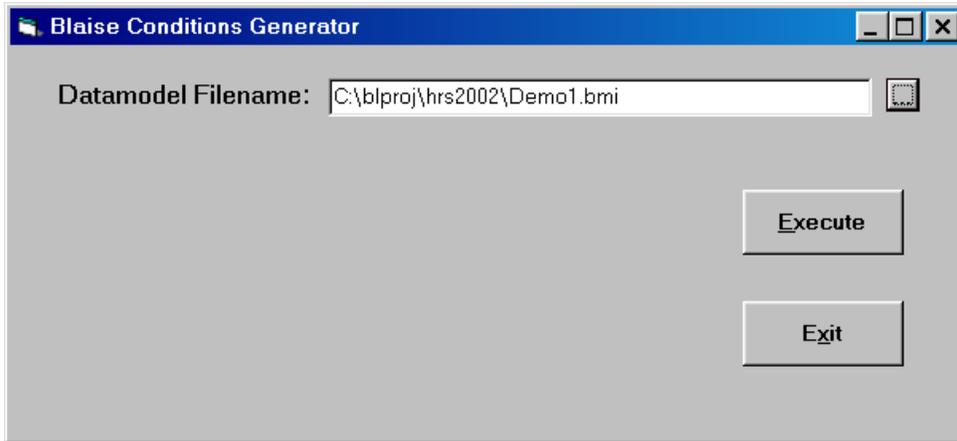
This is the screen first seen when running Universe.exe. You can type in a full path and datamodel name or use the select button to find your datamodel.



The Common Dialog box is opened and you can then select your file.



When a valid datamodel name is filled into the name field, all that is left to do is press the 'Execute' button and the condition file will be created. The variables table previously created by a Cameleon run will also be updated to reflect which fields are affected by what conditions.



This is the content of the condition file. The first item is the condition name (based on the first occurrence of a condition using field name) and the third item is the condition statements.

```
Name~Label~Condition
RWillig~~SYSDATE > TODATE (2000, 1, 1)
Driver~~(SYSDATE > TODATE (2000, 1, 1)) AND ((RWillig = Yes) OR (RWillig =
DONTKNOW))
CarTruck~~(SYSDATE > TODATE (2000, 1, 1)) AND ((RWillig = Yes) OR (RWillig =
DONTKNOW)) AND (Driver = Yes)
PetName~~HavePet = Yes
```

Note that only the unique conditions are listed.

The following is the contents of the variable file.

```
Name~Label~Object~Table~Condition~Location~Length~Type~Frame~Responses~Minimum~Maxi
mum~Decimals~Open?~DK?~Ref?~Empty?~Text
BlaiseKey~Blaise primary key~Demo1~Demo1~~1~0~Char~~1~~~~N~N~N~N~
Demo1Instance~Block Demo1 instance
number~Demo1~Demo1~~1~5~Numeric~~1~~~0~N~N~N~N~
RWillig~Is the respondant willing to take the
interview?~Demo1~Demo1~RWillig~6~1~Numeric~TYesNo~1~1~5~0~N~N~N~N~Is the
respondant willing to take the interview?
FavColor~What is your favorite
color?~Demo1~Demo1~Driver~12~20~Char~~1~~~~N~N~N~N~What is your favorite color?
HavePet~Do you have a
pet?~Demo1~Demo1~Driver~32~1~Numeric~TYesNo~1~1~5~0~N~N~N~N~Do you have a pet?
PetName~What is your Pet's
Name?~Demo1~Demo1~PetName~33~20~Char~~1~~~~N~N~N~N~What is your Pet's Name?
BlaiseKey~Blaise primary key~block1~block1~~1~0~Char~~1~~~~N~N~N~N~
block1Instance~Block block1 instance
number~block1~block1~~1~5~Numeric~~1~~~0~N~N~N~N~
Driver~Do you drive a car or
truck?~block1~block1~Driver~6~1~Numeric~TYesNo~1~1~5~0~N~N~N~N~Do you drive a car or
truck?
CarTruck~Car or
truck?~block1~block1~CarTruck~7~1~Numeric~CarTruck~1~1~2~0~N~N~N~N~Car or truck?
```

Future Plans

In the future, we plan to use more functions out of the BCP to replace the various pieces of information that we get from Cameleon currently such as field type, enumerations, loop levels, block names as well as all of the other attributes of the datamodel.

Internet interviewing using Blaise API

Bas Weerman, CentERdata, University of Tilburg

Abstract:

CentERdata is a survey research institute, specializing in Internet-based surveys. CentERdata is affiliated with Tilburg University, the Netherlands. CentERdata's most important activity is to carry out panel surveys through a telepanel: the CentERpanel. The CentERpanel is a panel of some 2500 households in the Netherlands. The members of those households fill in a questionnaire on the Internet every week. The CentERpanel is representative of the Dutch population.

In May 2000, CentERdata started doing Internet surveys using the Blaise for Internet system. It soon became clear that this system was not functioning optimally. CentERdata started to develop its own interview system based on the Blaise API calls. This system is now running for several months without any problems.

This paper will discuss (in short) our experience with Blaise for Internet and the possibilities of the Blaise API tools. It will describe the technical layout and the position of the Blaise API tools in our system.

Paper:

1. Introduction

CentERdata is a survey research institute, specializing in Internet-based surveys. CentERdata is affiliated with Tilburg University, the Netherlands. CentERdata's most important activity is to carry out panel surveys through a telepanel: the CentERpanel. The CentERpanel is a panel of some 2500 households in the Netherlands. The members of those households fill in a questionnaire on the Internet every week. The CentERpanel is representative of the Dutch population.

2. Conversion to a new system

In 1990, CentERdata started with a system of computer-assisted interviewing. All households had a small computer which was provided by CentERdata. With this computer the respondents phoned in to the CentERdata main server to receive their interviews. After the respondents filled in their questionnaires, they contacted the server again to deliver their answers. The system was built in-house by CentERdata programmers.

After 10 years it became clear that the interview system needed replacement. There were new requirements and the (now) old computers at the respondents' homes began to drop out and were not replaceable. In March 2000, CentERdata switched to a new system based on the Blaise for Internet package. The old system had been in use for almost 10 years, so the new system had to face a number of conditions:

- The system should look and feel almost the same as the old system. The conversion should not cause any problems for the respondents to fill in the questions (specially for the elderly).
- CentERdata needed an instrument before the actual questionnaire. This instrument was needed for identification of the respondents, by placing additional buttons on the screen for each respondent. Multiple buttons per respondents where possible (for more questionnaires per respondent). The Blaise questionnaire was then called with the appropriate household number and respondent number.

- CentERdata needed real-time logs of the users activities so online help could be provided.
- CentERdata uses setup boxes for panel members without computer. Those setup boxes don't support javascript, so all the intelligence should be on the server side.

In order to comply with those conditions, CentERdata decided to write a program between Blaise and the client browser. This program was called C2B (CentERdata to Blaise).

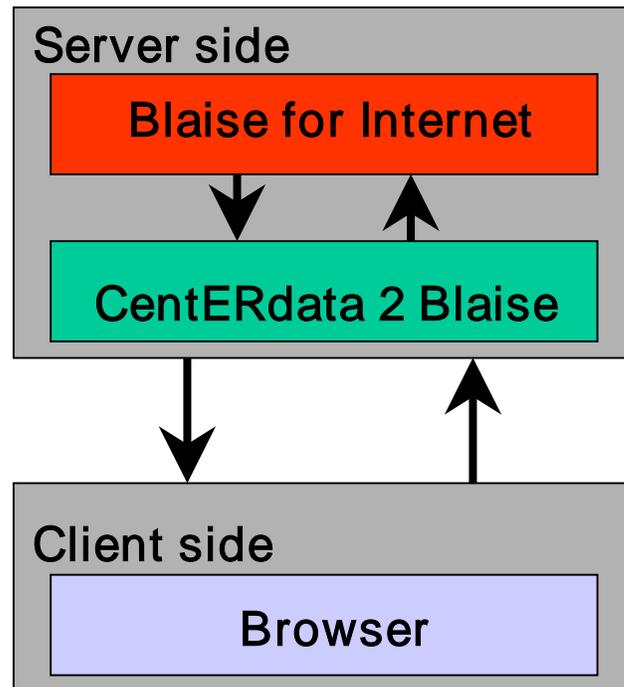


Figure 1: Layout C2B system

3. The layout of the C2B system

The C2B system was written in Delphi and installed at the server next to Blaise for Internet. The general structure of the program is as follows:

The communication between Blaise for Internet and C2B is completely in HTML. When a user logs in and requests an interview, C2B sends an html form to Blaise for Internet. The output of Blaise for Internet (in html) is transferred and sent to the browser. The question is shown in the client browser. The answer is sent back from the client browser to C2B, which transfers the HTML code with the answer from the browser into something understandable for Blaise (also in HTML code). There is no checking of the answer. A possible wrong answer can be sent to Blaise. If an answer is wrong (for example a range error), the output of Blaise is the same answer (because BIS didn't accept the wrong answer and in reply asks for the same question). C2B resends the question to the client browser, together with an error message. The error message is nothing more than: "There is an error in your answer or maybe you didn't answer."

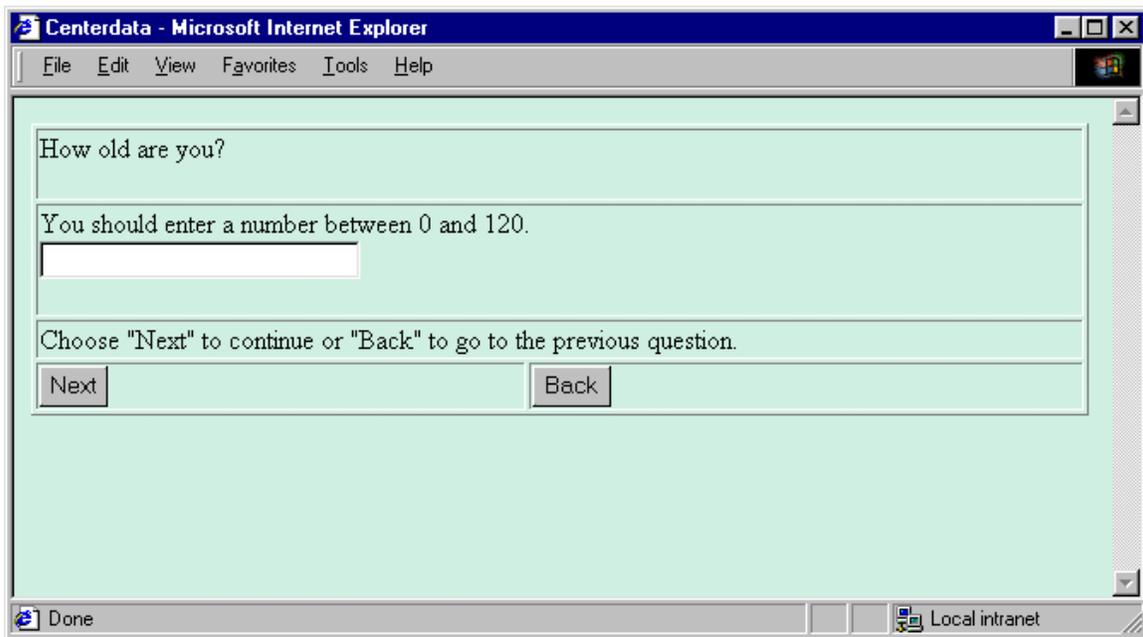
The Blaise for Internet package does not support tables. CentERdata needed the table support. In order to view tables on the screen, CentERdata has built a parser inside the C2B service. This parser gets the Blaise question text and looks for some special code. In this way, CentERdata has built a number of new features.

4. C2B features

C2BBack()

Normally, only a 'Next' button is placed on the screen, sometimes accompanied by a 'Refusal' or 'Don't know' button (if those options are assigned to the field). To add a 'Back' button, the text C2BBack() is added to the Blaise question text. When the C2B parser sees this code in the Blaise question text, it knows that a back button should be placed on the screen. The Blaise code looks like:

Question2 "How old are you? C2BBack() ": 0..120

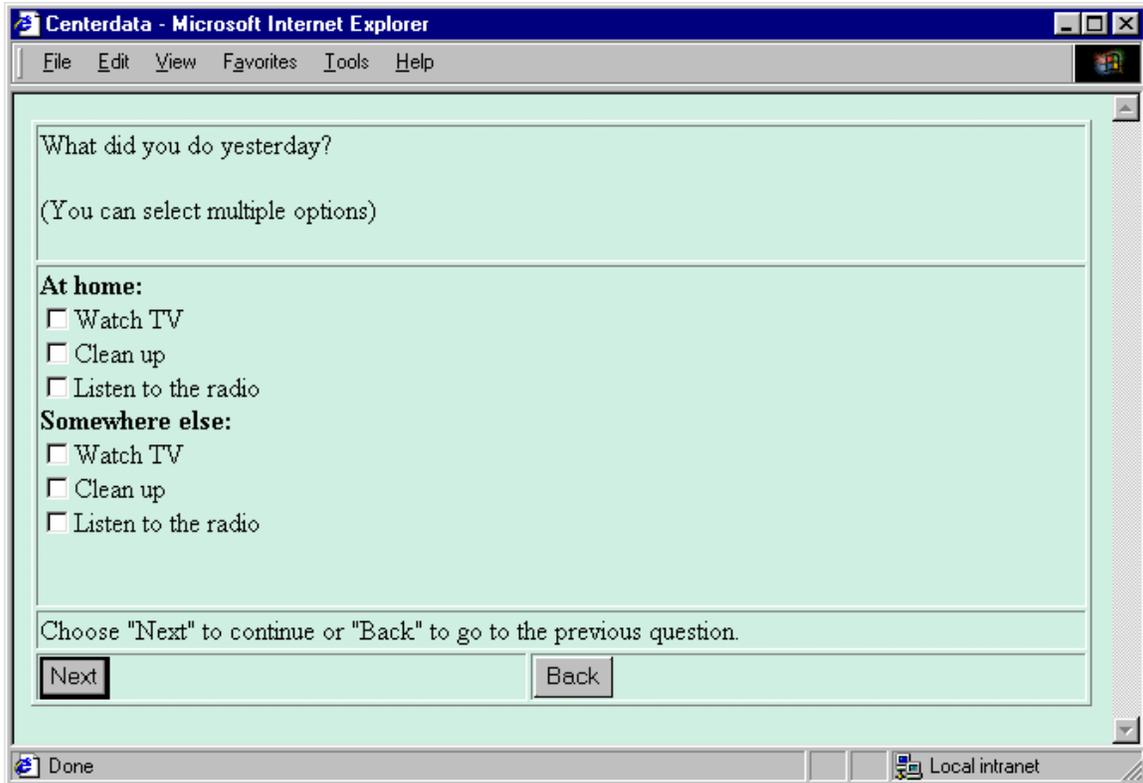


Screen 1: The use of C2BBack() in the question text

C2BNoOption()

Normally, all the answer possibilities of a set question are shown on the screen. To avoid a checkbox to appear before an option, it is possible to use C2BNoOption() in the set type.

Question3 "What did you do Yesterday?@/@/(You can select multiple options) C2BBack() ":
(
 a1 "@b At home @b C2BNoOption()",
 a2 "Watch TV",
 a3 "Clean up",
 a4 "Listen to the radio",
 a5 "@b Somewhere else:@b C2BNoOption()",
 a6 "Watch TV",
 a7 "Clean up",
 a8 "Listen to the radio")



Screen 2: The use of C2BNoOption() in the question text

C2BList()

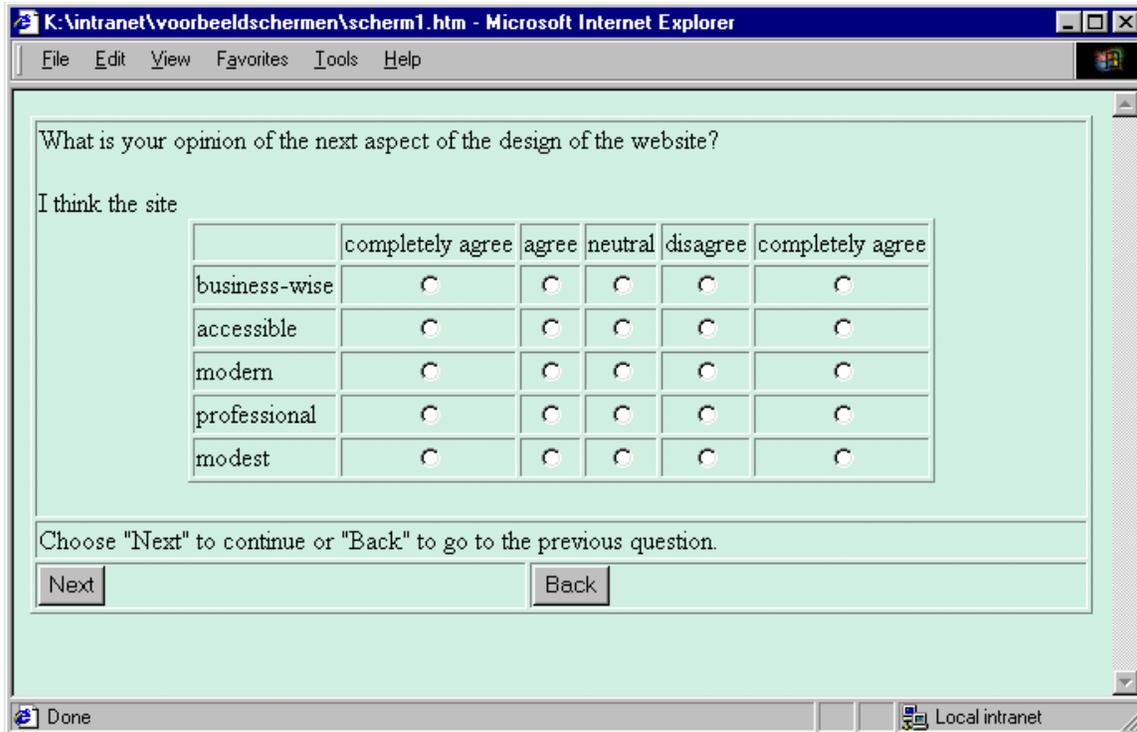
To support tables on the Internet, CentERdata has built the C2BList() option. With this option it is possible to place tables on the screen. The output of a question built with C2BList is of type String. The String is then transformed in the appropriate fields. This is done entirely in Blaise. Normally, the C2BList question could be of type Auxfield.

Fields

```
v091 "The website is business-wise? (o1 "completely agree", o2 "agree", o3
      "neutral", o4 "disagree", o5 "completely disagree")
{...}
Question4 " What is your opinion of the next aspect of the design of the website?
          @/@/I think the site
          C2BList(5, 5, 55555, 11111, "", "", 'completely agree', 'agree', 'neutral', 'disagree',
          'completely disagree', 'business-wise', 'accessible', 'modern', 'professional',
          'modest') C2BBack()": String
```

Rules

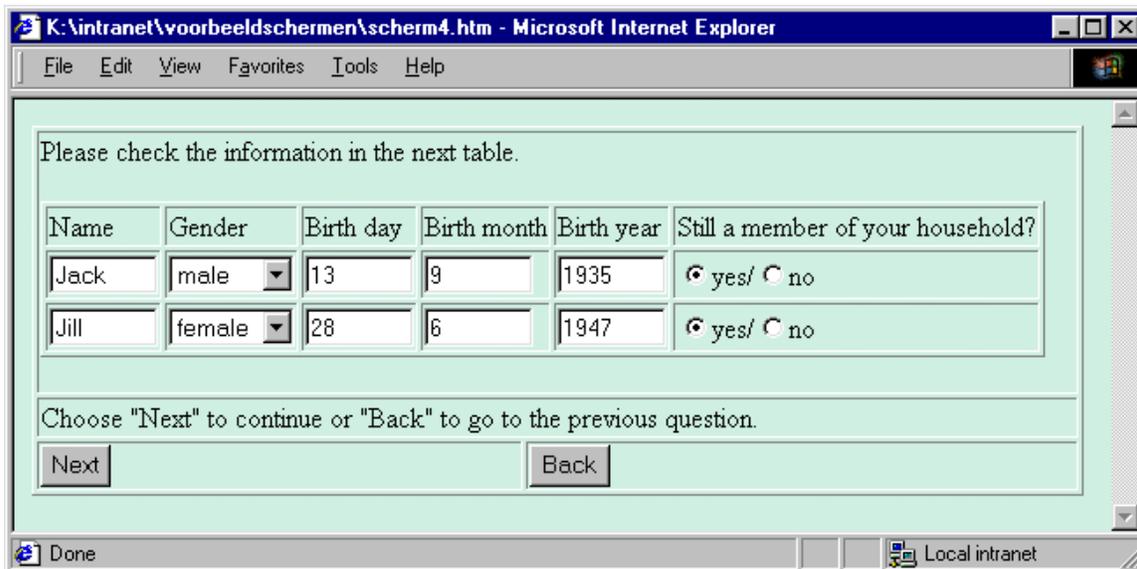
```
Question4
Str2Arr (Question4, 5, TempArray, right)
right = 1 INVOLVING(Question4) "You forgot to answer a question"
v091 := TempArray[1]
v092 := TempArray[2]
v093 := TempArray[3]
v094 := TempArray[4]
v095 := TempArray[5]
```



Screen 3: The use of C2BList () in the question text

The procedure Str2Arr in Blaise breaks the string into an array of integers. With this array, the original fields are filled. The procedure takes the original question text (the answer to Question 4), a number of expected answers (5), a TempArray which stores the array of integers and a right field which is filled with 1 if there are 5 answers given.

The use of C2BList is very advanced. Multiple answer types are supported, next to preload and advanced checking of the answers.

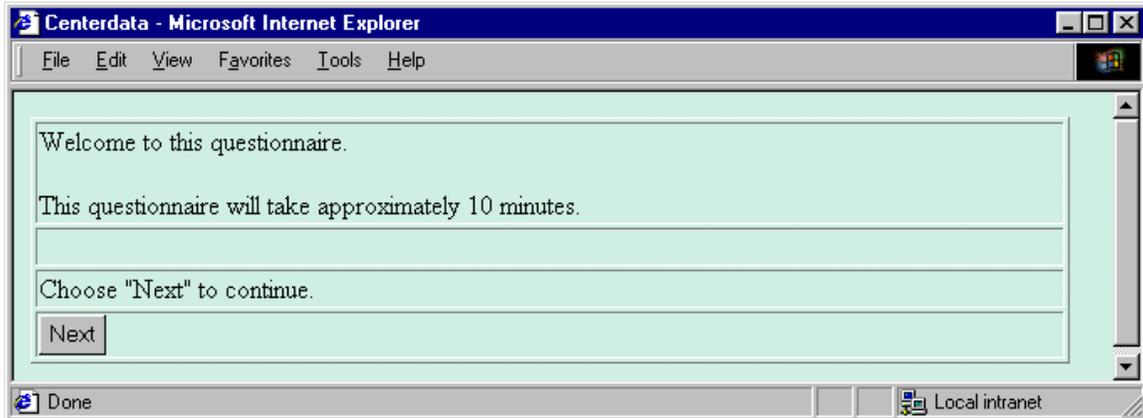


Screen 4: The use of C2BList() in the question text

C2BEmpty()

Sometimes a question is not really a question, but just used for information for the respondent. In that case it is possible to use C2BEmpty() in the Blaise question text. The normal input box is not shown.

Question2 "Welcome to this questionnaire. @/@@ This questionnaire will take approximately 10 minutes. C2BEmpty() ": String[1], empty



Screen 5: The use of C2BEmpty() in the question text

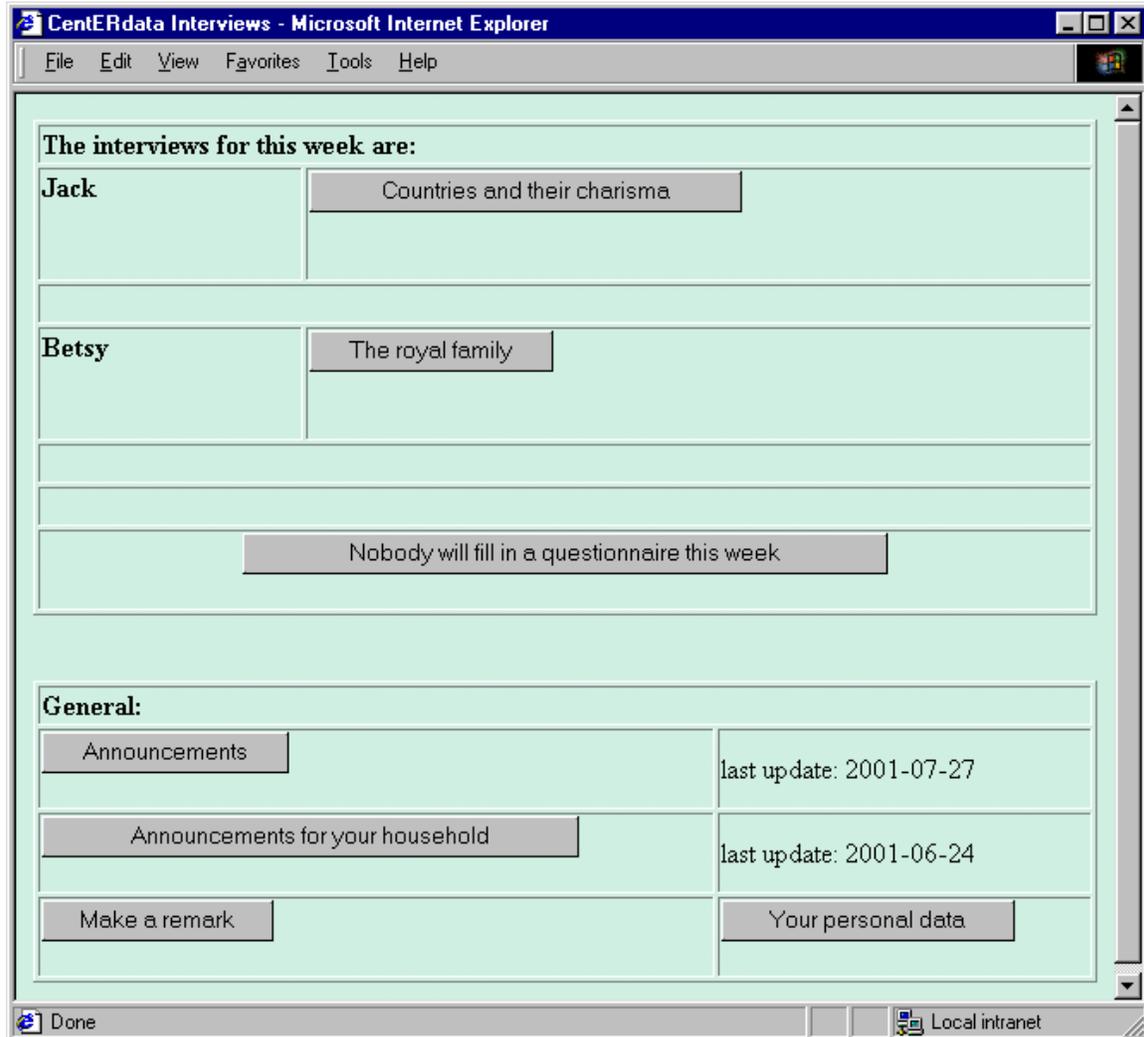
5. The interview registration system

Next to C2B, CentERdata has built an interview registration system. This system, built in PHP and MySQL, handles the administrative part of the interviews, the questionnaires to be presented to the respondents in the current week, the attempts of the respondents and the allowance for the telephone costs. When the user first logs on to the system, the next screen will be shown:



Screen 6: Logging on to the CentERdata interview system

Once the user enters the household number and password (which are sent by regular mail), the login screen disappears and the interview start page is shown. The next time the household logs on, the interview start page will be shown directly, because the household number is stored on the local machine. At the interview start page, the interview start page is shown. Here all the members of the household are listed, together with buttons for the interviews for the week. It is also possible to make remarks and view announcements. The personal data (the allowance) is behind the 'Your personal data' button.



Screen 7: The start screen of the CentERdata system

It is clear that Blaise for Internet is not capable of handling administrative data like this. After clicking on an interview button, a screen is shown with additional information about this interview and a possibility to go back to the interview screen (in case a respondent accidentally presses the wrong interview button). After this screen, a PHP script (interview.php) is called which handles the communication with the client browser and C2B. This script is necessary to ensure browser independency. The post information is transformed into a standard input line which is sent to C2B. In this way, C2B does not have to worry about different browsers.

In C2B a list is created with all the current users. A unique ID is created with the household and member number. Every unique ID has its own object, which handles the communication with Blaise for Internet. If the ID is not yet in the list, it is created and the start interview command is sent in HTML form to Blaise. The response of Blaise for Internet is analyzed and if it contains a *nohouse* or *nomem*, the appropriate household number and member number are sent back to Blaise for Internet (in HTML format). After this procedure, the actual interview is started. The respondent never sees that the first two questions are answered and gets the first question on the screen the moment the start interview button is clicked.

For every question, the respondent has 20 minutes to answer. If this time has passed, the quit interview command is sent to Blaise in HTML format. Internally the Blaise timeout is set to 30 minutes, so C2B is always earlier than Blaise for Internet with the timeout command. If the respondent wants to continue the interview, he has to start all over again, because Blaise for Internet only supports the New command.

6. Problems using Blaise for Internet

CentERdata started using the new system in March 2000. After a few weeks it became clear that there were a few problems concerning Blaise for Internet.

Problems regarding the content

Blaise for Internet does not support the 'get' method. Every time a questionnaire is started, a new record in the database is created. If a respondent wants to stop the questionnaire and continue at a later time, a new record is created. The questionnaire starts again from the beginning. This problem caused a lot of inconvenience for the respondents. A lot of questionnaires had to start all over again because the server crashed several times during the weekend, because of Blaise for Internet. This problem was presented at Statistics Netherlands. A new version of Blaise for Internet was developed. This version supports the 'get' method, but this version turned out to be so unstable that CentERdata never used or tested it.

Another problem is the table support. Blaise for Internet does not support tables. CentERdata had to develop its own table support by creating a parser in between Blaise for Internet and the Client browser. The development of this system and the training of the Blaise programmers was very time-consuming.

Technical problems

The first version of the Blaise for Internet system was not capable of handling more than 10 respondents. After the 10th respondent, the system started to assign wrong unique numbers. The result was that respondents started to fill in questions for other respondents and were getting the information meant for other respondents. Statistics Netherlands quickly acknowledged and solved this problem.

The Blaise for Internet system was delivered as an add-on to the Microsoft Information Server. The whole system turned out to be very unstable. The only solution to a hang up, was a hard reboot of the server. In some weekends, the server was (automatically) rebooted over 20 times. In the beginning the workload of the system was approximately 45 respondents filling in questionnaires at the same time. The maximum number of interviews filled in during the weekend was 4000. In a few months, CentERdata started to use the system for the whole panel. The upper limit of 50 respondents working on the same Blaise for Internet was quickly reached. CentERdata installed a second server. C2B redirected the respondents to the second server when the limit of 50 was reached. Later on, Statistics Netherlands changed the upper limit of the concatenating users.

In spite of the full cooperation of Statistics Netherlands, it soon became clear that this system was not working for CentERdata. A lot of respondents decided to quit their association with the panel due to the technical problems. CentERdata decided to build its own interview system based on the Blaise API calls.

7. The new CentERdata 2 Browser service

After all the effort invested in the new system, it was decided to use the old C2B service as a base for the new system. Instead of the communication with Blaise through HTML code, it would be much easier if it were possible to communicate directly with the Blaise database and metadata. Fortunately, the beta version of the Blaise API calls was available for testing. The API calls made it possible to connect directly to the Blaise database. After all, CentERdata only extracted the question text and answer possibilities from the HTML code Blaise for Internet generated. With the API calls it is possible to extract the question text and answer possibilities from a Blaise database.

8. The layout of the new CentERdata 2 Browser service

The new CentERdata 2 Browser system now also decides which question is shown on the screen. An overview is shown below.

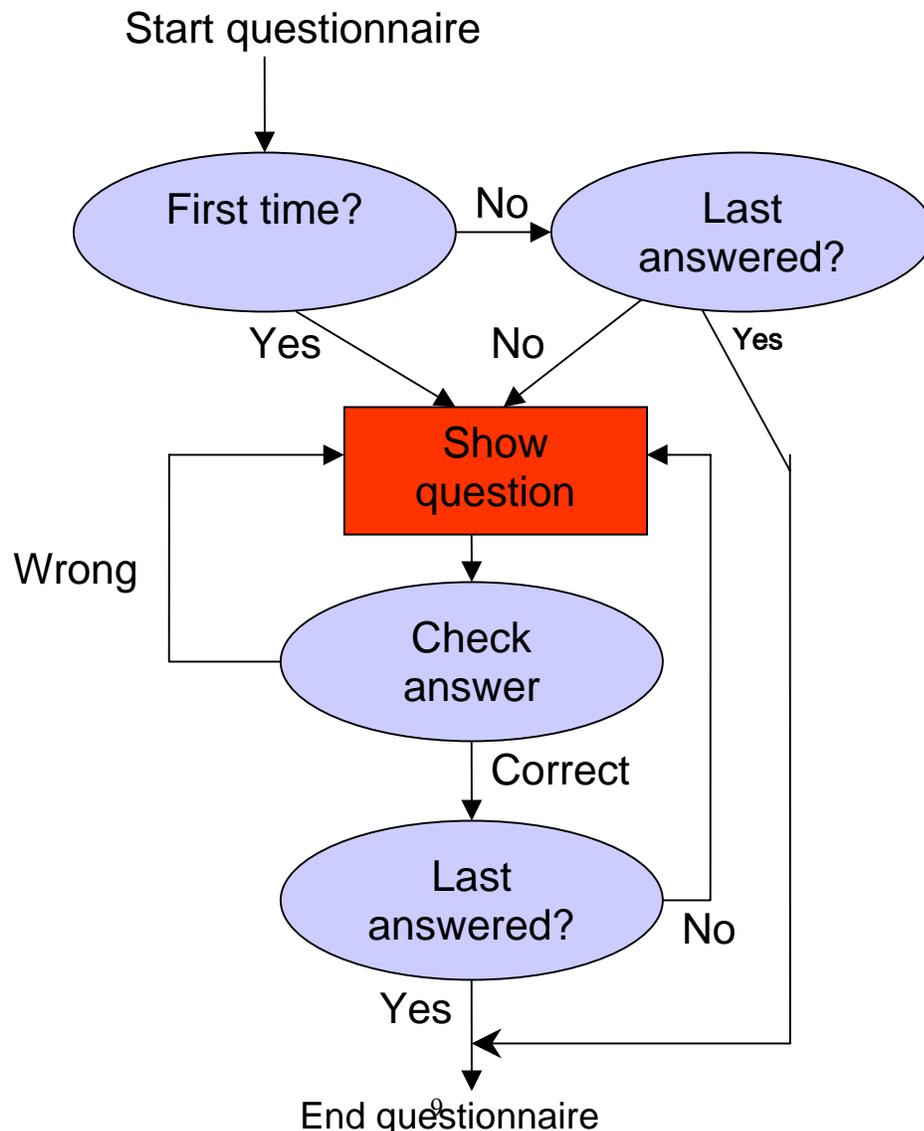


Figure 2: An overview of the Show question decision

If a respondent starts an interview, a new entry in the respondent list is made. If the respondent is already in the list, he continues with the last question. Every respondent stays in the active list for about 30 minutes. After 30 minutes of inactivity, the respondent is deleted, and the next time, a new entry should be made in the respondent list.

After the registration in the respondent list, a 'get' command is sent through the Blaise API calls to the database based on the household and member number. If this command fails, a 'new' command is sent, creating a new record for this respondent. In case of a 'new' command, the first question is shown on the screen (together with some additional information like 'You should enter a number between 0..100'). If the 'get' method is successful, the 'getlastquestion' command is sent, and the first question the respondent has to answer is presented. The respondent is now capable of continuing a questionnaire.

Once a respondent has answered a question, a 'checkanswer' command is sent through the API calls. If 'checkanswer' fails, the question is resent to the client browser, together with an error message ('You should enter a number between 0 and 120.'). If the 'checkanswer' command accepts the answer, the answer is stored and the next question is shown on the screen.

This new system supports all the Blaise for Internet features. The programmers at CentERdata did not need to change the way they were programming. The new C2Bservice based on the Blaise API still used all the C2B commands. Tables are shown question by question.

9. C2B with API calls in practice

Since the complete transition to the new C2B system in January 2001, CentERdata has few to no technical problems. The number of times the server crashed is reduced to almost zero in a normal interview weekend from Friday till Wednesday. The possibility for respondents to continue their questionnaire has proved to be very powerful. The CentERpanel has grown to around 3000 households. The maximum workload so far is around 110 panel member filling in a questionnaire at the same time. The maximum number of questionnaires handled in one weekend is 10,000.

10. Future developments and hosting of Blaise questionnaires

In the near future CentERdata will try to be fully Blaise compatible. We are now working on the implementation of the tables.

From January 2001 CentERdata started to host Blaise questionnaires on its server. The results so far are very promising. Also a few other institutes are testing C2B on their own server. The C2B service is very simple. There is no intelligence in the client browser. The error checking is completely done on the server side. There is no need to reinvent the Blaise system in Java or XML. C2B service is browser-independent. The layout part of C2B is placed in a separate database. It is possible to generate a different layout for every questionnaire or even every question in an interview.

11. Conclusion

Although CentERdata faced a lot of problems during the conversion of its old interview system to the new Blaise-based system, the Blaise system behind C2B proved to be a major success. The incapability of Statistics Netherlands to solve a number of essential bugs in the Blaise for Internet package was not caused by incompetence or a lack of interest. CentERdata was just too far to the edge. The Blaise component pack is a huge step forward for everybody who wants to control Blaise and does not want to be dependent on the Blaise development team.

Session 8: Audit Trails and Testing

- Testing a Production Blaise CATI Instrument
Amanda F. Sardenburg, U. S. Bureau of the Census
- The Practical Application of Audit Trails: A Case Study
Rob Bumpstead, Office for National Statistics, United Kingdom
- Automated Testing of Blaise Questionnaires
Jay R. Levinsohn and Gilbert Rodriguez
Research Triangle Institute, USA
- Reporting on Item Times and Keystrokes from Blaise Audit Trails
Sue Ellen Hansen and Theresa Marvin
University of Michigan, USA

Testing a Production Blaise Computer-Assisted Telephone (CATI) Instrument

Amanda F. Sardenberg, U.S. Census Bureau
John W. Gloster, U.S. Census Bureau

Abstract

Good testing practices are just as important to a Blaise computer-assisted telephone interview (CATI) system as they are to any other production software system. A rigorous approach to testing can put the system in production sooner and avoid production setbacks in the introduction of enhancements. We will focus on the Blaise CATI system, which supports the Telephone Followup (TFU) operation of the American Community Survey (ACS). The ACS generates over 10,000 problem cases a month for TFU. It is imperative that this system function efficiently and as close to error-free as possible. Careful testing is a key ingredient to making this possible.

We will address the following aspects of our testing approach:

- ~ Maintained detailed specification: The many specialized aspects of Blaise software can require the same level of complex, lengthy specifications and production testing as a much larger system. Instrument optimization testing would ideally account for every possible path or universe, every possible error condition and flag, and every possible coded outcome prior to each new production release. This level of testing requires detailed specifications and rigorous specifications maintenance.
- ~ Separate testing environment: Staging grounds, that is, a test environment, must be established in order to exactly mimic the production environment; additionally, if the test environment must be accessible by multiple users other than the administrator, a pre-test environment would also need to be established.
- ~ Test plan: The different testing paradigms and designs, balanced usage and cyclic versus continuous implementation of the respective testing procedures will be explored in more detail using actual experiences garnered while conducting ongoing testing for the American Community Surveys Telephone Follow-Up (TFU) Blaise CATI instrument.
- ~ Test log: Handling priority items of this nature is most efficiently done in an inclusive, communal manner involving administrative, analytic, and developer staff. For this purpose, a highly detailed tracking log is maintained by the administrators while incorporating analytic tester results and input from developers.
- ~ Version control: Procedures involving date stamps and structured network storage of code delivery, archived or frozen source code, current data and metadata, etc., also need to be established and followed rigorously for ease of maintenance, tracing, and reinstatement of code, if ever needed.
- ~ Production simulation: Testing must replicate the production environment in order to accurately assess the viability current instrument fixes or upgrades. Ongoing modifications to a production system also mandate other, more practical considerations that must be balanced with any thorough, systemic specifications and modifications testing.
- ~ Deployment of new releases of the Blaise instrument: All optimization testing needs to simultaneously be balanced with other efficiency constraints, such as staffing, time to next upgrade of software or instrument, and issue priority. Additionally, new releases of Blaise software require their own testing to assess their fitness in the production environment
- ~ Diagnosis of and response to problems in the production system: Optimization testing during

development and modification of a Blaise instrument will often reveal pressing issues that must be resolved to maintain minimum required functionality. We have implemented an issues and problems tracking log for speedy resolution.

The ACS will expand in the next few years by a factor of 3.5 over current workload. Testing of the Blaise CATI TFU system will take on an even greater importance with this demanding workload.

Introduction

The American Community Survey (ACS) was begun for the purpose of collecting current demographic information on housing and population data on a continuous basis. The ACS consists of a mailed questionnaire with an automated clerical edit for questionnaires. The initial by-hand determination of what was sent to TFU was automated via a set of algorithms that determine whether a survey passes clerical edit (CEDIT) or fails and is routed to TFU. The automated telephone follow-up (TFU) operation that utilizes Blaise software, and a non-response CATI and computer-assisted personal interview (CAPI) component, round out the survey operations. Within the next few years, the U.S. Census Bureau's American Community Survey (ACS) is slated to become the largest permanent and ongoing survey in the country, sampling approximately 3 million households annually.

Thus, the process of automating TFU became critical as the geographic scope of the survey expanded from a small pool of test sites to using the ACS methodology currently in 1,239 counties across the country, and the time from conception to initial production operation was extremely brief. As a result, the TFU instrument was first operational in an essentially beta form and has required extensive and continual testing to maintain its production functionality and data-handling integrity. Ongoing testing has remained a critical aspect of the maintenance and periodic upgrade of our production instrument.

Specification Maintenance and Testing

The many specialized aspects of Blaise software can require the same level of complex specifications and production testing as much larger systems, regardless of sample size and scope. Ideally, instrument optimization testing will account for every possible coded outcome, path or universe, error condition, or flag prior to each new production release or software upgrade. This level of testing requires detailed specifications that are rigorously maintained and are then used to then develop testing procedures and to baseline the instrument as modifications are introduced. Once the instrument specifications have reached a certain level of development and have been reliably translated into stable instrument functionality, these essential specification verification routines would ideally become automated.

Maintenance of Separate Environments

A production instrument in a remote location mandates that the tester be able to replicate that environment exactly. This is necessary to accurately assess the viability of current fixes or upgrades. Ideally this environment is local and is physically separate from the actual production environment. Staging grounds, that is, a separate test environment, must be established in order to exactly mimic the production environment while testing new code or new functionality. Additionally, if the test environment must be accessed by multiple users, additional pre-test environment(s) would also need to be established. This allows for testing the basic integrity of new code deliveries or performing other

administrative and environmental management tasks (or other more selective or disruptive instrument testing) without disturbing routine or ongoing testing. One example of an additional environment might be one maintained strictly for training. Another might be one maintained for certain disruptive tasks, such as testing that involves the Blaise scheduler and the appointment block, or for extensive recompiling of code.

Test Plan

Experience has shown testing to fall broadly into two major categories: core testing and “ad hoc” testing of specific issues or problems, typically arising from either routine testing or routine interviewer usage of the instrument. Core instrument specifications must be identified for core testing.

Core testing is both periodic and ongoing, as needed. It is typically done prior to deployment of new TFU instrument releases and also prior to any new upgrades or conversions to new Blaise software releases. It may also frequently be performed in the interim of either of the two situations above, based on implementation of new spec changes or new code deliveries, or based on problems that may have been detected. Examples of core testing for our TFU instrument include all types of essential specifications testing listed below:

1. Verification of data integrity and case disposition concerning data read into and out of DEP, and as it appears in the datafile, call history, audit trails, etc., per read in, read out, clerical edit and other specifications
2. Verification of pathing and universe constraints against the datamodel specification
3. Verification of correct outcomes, agendums per cumulative outcome, numberroute, and dial counter specifications
4. Verification of essential header text, question text, and both response categories and response text against questionnaire specifications
5. Verification of screen layout of infopanels, formpanels per question per the front/back specifications

“Ad hoc” item testing is ongoing and is based on any outstanding issue or problem that has been reported. A dated and detailed testing log is maintained to describe and track these problems as they arise and will also be described in more detail below. The implementation of both core and ad hoc testing procedures will be discussed below.

Implementation and Usage of Blaise Instrument Testing Plan

To maximize office resources and develop an efficient system for approving changes to Blaise code, various testing plans and procedures are employed for the Blaise TFU instrument. The testing plans are geared to cover a variety of instrument changes, including but not limited to: structure changes, data input routines, questionnaire content, and outcome code assignments.

During the course of ACS survey management, changes to the daily operation may require instrument specifications to be revised and updated. To update the Blaise TFU instrument, requests and revised specifications are submitted to in-house and/or contract developers for recompile. Once the new code is received, it is incorporated in a recompiled Blaise TFU Data Entry Program for testing. Unfortunately, with multiple change requests for a particular block of code, any lapse in strict version control may result in reintroduction of outdated code into the instrument. While performing core testing or cursory checks, aberrations or unexpected outcomes are invariably identified. These aberrations, such as improper skip

patterns, outcome codes, or incorrect subject or verb fill, are typically related to the most recent changes but also may have existed for an indeterminate amount of time before being identified. Consistent usage of regular testing approaches is the best way to identify and resolve aberrations.

When a new version or build of Blaise software is released, our organization decides whether or not to upgrade our current Blaise survey instruments based on evaluation of expected benefits to survey operations, and/or on recommendations from Westat (the North American Blaise technical representative) or our systems technology staff. At other times, significant changes to the TFU instrument structure are made specifically to elicit desired instrument behavior or resolve particular issues. These two scenarios, in addition to the circumstances described above, require varying degrees of testing. Detailed below are aspects of our testing plan that are employed, dependent upon the circumstance and level of instrument change.

Cyclical and Continuous Testing

Whenever an instrument specification is changed and Blaise code is subsequently modified to accommodate the new spec, a thorough testing of the instrument portion must be performed. Once the new code is recompiled, TFU instrument testers conduct several mock interviews in the data entry program (DEP) to check that the instrument was modified correctly (to specification), and that other areas of the instrument were not inadvertently affected as a result of the code change. For example, modifying a block of code without regard to its existing parallel code in a separate file would cause inconsistent instrument behavior. If the initial instrument problem was not specification-related (for example, a problem discovered while testing another modification or problem), the same type of testing is administered once the problem is identified.

Once the instrument change has been successfully tested, that version is maintained in the separate test environment until it is deployed in the production environment (see Maintenance of Separate Environments, above).

Troubleshooting and Ad Hoc Testing

At times, we are unaware of instrument problems unless we are informed of TFU-related production problems by the staff in ACS TFU Unit of the Census Bureau's National Processing Center in Jeffersonville, Indiana. Reliable lines of communication are fostered to facilitate this exchange of information.

Once we are given details about the instrument behavior, preferably with specific examples, testers at Headquarters then try to replicate the problem based on the information provided. This usually consists of specific keystrokes performed within particular types of cases to produce desired outcomes or pathing sequences (whether correct or incorrect). If a certain case is referenced where the problem was discovered, we have the ability to reference these particular cases directly for a given number of days within our Blaise production database archives. This further helps us to identify and emulate as much as the possible the environment(s) within which the problem was first recognized. If the problem is indeed instrument-based (which is the norm) and not interviewer-based, a request is made to our developers to rectify the code, where we would then in turn re-test to verify that the problem has been resolved. Interviewer-based issues may require adjustment to training procedures, which are handled elsewhere.

In similar respects, there also are problems that are discovered in-house that are handled in the same manner. Replication of errors and careful documentation is always performed before making requests of developers to alter code.

Corel WordPerfect - S:\fu\bug\testers_paper\testing_numberroute sample.wpd (unmodified)

File Edit View Insert Format Tools Window Help

Times New Roman 11 B I U Table General

IF (DIAL1.D111=1 OR 5 and MAIN.RESPNAME <> empty) or DIAL1.D1112=1, Do not increment MAIN.ATELCTR or any DIALING COUNTERS. Set outcomes without incrementing the counters so that you do not move onto the next phone number....
Triggers and rules for updating MAIN.UTEL and setting associated variables
After setting the appropriate Counters, UTEL, Outcome, and Agenda reset the trigger Variable to <blank>

Conditions (Edit)				Actions				If Correct Outcome Assigned? Y/N (If N note Adjunctive)	Does Counter Increment Correctly?	If Counter Does Counter Edit Succeed?
Universe /Trigger	Counter Name	Counter =/>	ADMIN COUNTERS > ADMIN ATELCTR	Increment	Reinitialize all Dialing Counters	MAIN UTEL	CALL ANALYSIS Outcome	Go to or Route to		
DIAL1.D1112=2			Y	MAIN.ATELCTR		MAIN.ATEL(ATELCTR)	57	DIAL1.ACS_DIAL		
			N				22	MAIN.NOTES		
PROBLEM PROBLEM=3			Y			MAIN.ATEL(ATELCTR)	59	DIAL1.ACS_DIAL		
			N				13	MAIN.NOTES		
DIAL7.D71=1			Y	MAIN.ATELCTR		MAIN.ATEL(ATELCTR)	52	DIAL1.ACS_DIAL		
			N				124	MAIN.NOTES		
DIAL7.D71=2	DIAL7.DCTR	Y	Y	MAIN.ATELCTR	DIAL7.DCTR RESA. RESPONSE SPIC FAX.FASCCTR BADCONNECTOR.BCCTR	MAIN.ATEL(ATELCTR)	52	DIAL1.ACS_DIAL		
		Y	N				124	MAIN.NOTES		
		N		DIAL7.DCTR		reinitMAIN.UTEL	55	MAIN.NOTES		

Document2 2001_tlog_07262001.wpd testing_numberroute s... AB TABLE A Cell A1* Pg 1 Ln 2.2" Pos 0.458"

Figure 1. Sample Page of Numberroute Specifications Testing

Core Testing

Several circumstances require intensive, detailed testing procedures to ensure all aspects of the instrument are functioning correctly. Among these circumstances are: Blaise version upgrades, remote and on-site instrument deployments, and any instrument structure changes.

At a minimum, instrument upgrades involve the core testing of the five specification and data integrity verification activities outlined above in the core testing Test Plan. Refer to Figure 1 (above) for a sample

testing document for Numberroute specifications. Additions to or expansion of current core testing areas may be incorporated as time and resources permit and circumstances require.

For version upgrades, a separate test environment is first upgraded with the new Blaise software version. Core testing is conducted by several testers, to divide responsibility (as some testers are more familiar with relative parts of the instrument), cross-check as needed, and minimize individual testing burden.

Other data integrity tests are conducted on, for example, Blaise input file and Manipula routines by computer specialists, to make sure that all data are being read in properly. Checklists and testing tables based on core testing specifications are developed to insure that testing procedures are completely thorough. Also, core testing is conducted whenever plans are made to deploy an updated TFU instrument. This helps to ensure that what is incorporated into the production environment is fundamentally sound.

An abbreviated version of the core testing procedures is performed for situations such as: limited instrument enhancements (for example, updates to the appointment block, or to CEDIT case disposition algorithms), and code modifications that may affect other blocks in instrument (for example, cumulative outcome table updates and revisions to core instrument [acsform.bla] code). Depending upon the degree of instrument change, the abbreviated core testing may range from four core activities to as few as two. When it is determined that an exhaustive test is not necessary, the core testing is refined to focus on only the essential areas of concern.

Testing Log

To maintain an efficient method of organizing each testing occurrence, we developed a testing log (Tlog) which we maintain and circulate to everyone involved with the TFU operational maintenance. The Tlog is essentially a running historical table that documents the following information: when an instrument problem is first discovered or reported to us, a detailed description of the problem, its correction status, to whom it is assigned, and whether or not the problem has been fixed.

Planned Enhancements to Testing Procedures

Although our testing plan is relatively standardized, it still lacks systematic methods for testing certain code fixes. As mentioned earlier, our experiences testing the Blaise TFU instrument have underscored the fact that code modifications may inadvertently affect other instrument functions. Central to planned enhancements to current testing procedures is the development of a TFU Testing Checklist that testers can reference and utilize in every testing scenario. Certain code fixes (for example, changing an error flag value) will be assigned specific instrument functions which are dependent upon the code fix (for example, checking active signals, error status blocks, and relevant LookSee fields). Instead of having to know or determine which instrument functions to test for a specific code fix, a tester will be able to refer to a complete listing of which aspects of the instrument are involved for all routine core testing situations. As the ACS operation expands in 2003, so will the number of cases that become eligible for TFU. A testing checklist will help conserve resources, minimize testing time, and help to ensure that the proper functions are being covered in the testing.

Testing Log

To maintain an efficient method of organizing each testing occurrence, we developed a testing log (Tlog) which we circulate to everyone involved with the TFU operational maintenance. As mentioned, the Tlog is essentially a running historical table that documents the following information: when an instrument problem is discovered (or reported to us), a detailed description of the problem, its correction status, to whom it is assigned, and whether or not the problem has been fixed.

Blaise Data Entry - i:\mfudata\testdata\blaisedata\vacafom

Forms Answer Navigate Options Help

ACS Appointment1 CurrentStatus1 ChangeRespondent1 FAQsGenHelp1 BadConnection1

CATI mode. CMID = 018220056. MAILREC = 1. RDATE = 2/7/2001. SECTION = P7
—MAVIS M BRACKIN**MAVIS M BRACKIN. Gender =F. Date of Birth = 12/10/1934. Age = 66.
Relationship= —
During the week of 1/31/2001,
were you TEMPORARILY absent from a job
or business?

1. Yes, on vacation, temporary
illness, labor dispute, etc.

2. No

	NWLA	2	No
	NWAB	1	Yes
	NWRE	2	No
	NWLK		
	NWAV		

Old 29/85 Modified Dirty Navigate ACS

Ex.1

When first entering a 1 in NWLA, then a 2 in NWRE - this correctly brings NWLK on path. I then back up and change NWLA to 2, which correctly brings NWAB on path. When NWAB = 1 the instrument should go to WKL. Instead it goes to NWLK which remained on path because of the NWRE 'keep'= 2.

Figure 2. Sample Diagnostic Graphic for Troubleshooting

Each instrument problem is given a unique sequence number to easily refer to the item. If a fix of the item is attempted but not sufficient, the testing result is dated and documented for that item.

This document serves as an excellent reference for all concerned parties and is easily disseminated upon request. Newly added Tlog items are given a priority number of 1, 2, or 3 (with 1 being the highest priority) to help the developers prioritize their code modifications. For instance, an item that is not detrimental to the success of a TFU interview, such as certain infopane typographical errors, could be given a priority 2 or 3, whereas an item that details an improper skip pattern would be given a 1. It is also possible to give higher priority to otherwise less important items if an instrument deployment is planned and all relevant Tlog items must be completed or fixed by a certain date.

At times, the statement of an instrument problem in the Tlog may not be clear enough for a developer to understand. Often testers and developers discuss a problem or may even meet so that the tester can demonstrate the problem or provide supplemental information to further clarify the problem. Figure 2, above, is a sample diagnostic attachment from a mail message sent from a tester to a developer.

TFU Instrument										
Line	Block	ITEM	Description of Problem/Change	Reported	Resp.	Priority	Tested	Passed TMDt	Passed TEST	
384	Housing cont.	vacant HUs	When a unit is declared vacant in the census uses block, the Housing block automatically comes on path. Depending on the type of vacancy indicated, only certain Housing block items are required to be answered, thus taking unneeded Housing items off path regardless if they are filled or <Empty>. When an interviewer has answered all relative Housing questions, the Housing cont. active signal comes up if one or more of path Housing items are <Empty> (even though the block was never "skipped"). Please correct so that the active signal only comes up if the block is skipped (only possible during non-vacant interviews).	6/12/01 KPS, KC	JG	2	6/27/01	N/A	6/27/01	
384b	Housing	Current Status	Housing error counts are adjusted depending on the type of vacancy. This doesn't always happen correctly... for example: CID019978602 - Crit Housing counts = 4 though no errors are flagged for an active signal since all pertinent items are answered (goes to Finish). 7.1301 example CID# 02002368 - upon entry into the case, CritH = 6, MedH = 0, LowH = 4 (although these values were 7,0,2 at MakeDtg). - in the LookSee fields the flagged items were -> ELE, WAT, TAK, INS, MRGX, SMX, GAS, FUL, YALL, and SR. - once CJIC2 = 2 for vacant unit, the CurrentStatus = 4,0,1 for TAK, INS, MRGX, SMX, YALL (correct) - once CJIC3 = 1 for For Rent, the CurrentStatus = 5,0,3 (almost correct) though LookSee had 6,0,3 -> RNT, RNTM, RNTS, PHP, YALL, TAK, INS, MRGX, SMX (the last 5 items should not be flagged due to renter status). - when in the Housing block, "page 3" (the Rent portion of the Housing block) came on path since all relevant item in pages 1 and 2 had already been answered. once items are answered in page 3, the instrument correctly routes to DorseHdon (end of page 5) and skips over page 4 (tax and mortgage info), though the counts are off due to the additional flagged items.	6/27/01 JWG	JG	2				
385	P7	PMM02, PMM04, PMM05, PMM06	Subject /verb disagreement has been reinstated.	6/11/01 KC, JWG	JMO	2	6/12/01		6/12/01	
386	P7	JMTR, JMTR1, JMLH, JMLH1	Subject /verb disagreement has been reinstated.	6/11/01 KC, JWG	JMO	2	6/12/01		6/12/01	
387	P7	INX1, INM2	Subject /verb disagreement has been reinstated.	6/11/01 KC, JWG	JMO	2	6/12/01		6/12/01	

Figure 3. Sample Page from Tester's Log (Tlog)

The Tlog has been invaluable in helping us document testing progress. It also serves as an archival reference for identifying new or recurring problems which may be similar to past instrument problems. Above is a sample page from our Tlog (Figure 3).

Version Control

Standard version control procedures involve nested folder locations, naming conventions, date-time stamping, and archiving of source code and data, regardless of whether manually done or whether versioning library software is used. This allows for ease of maintenance; code delivery tracking; comparison and troubleshooting of code, data, and instrument behavior; as well as for version rollback, should that be necessary.

In order to trace any versioning issues with current and newly incorporated code, we have found it necessary to maintain an orderly and structured network storage of code delivery and implementation. This has also allowed us to maintain a continuously updated historical archive of code. Additionally, analysis of dated code archives in conjunction with dated Tlogs has allowed us to evaluate past issues and solutions that were or were not utilized successfully for those issues with current issues that may be identical or similar. This has proved invaluable on a number of occasions, particularly as the issues and problems have become more complex.

Deployment

There are two major types of deployments: instrument upgrades and software upgrades. Ideally these two types of deployment are done separately in order to avoid conflating sources of possible problems or outcomes on the production instrument.

All instrument upgrades require regular core testing, as mentioned earlier. For Blaise software upgrades, in addition to those tests described earlier, this core testing is typically augmented by a variety of other system-level tests prior to deployment. This combination of internal and external testing is necessary to assess fitness in the current production environment before implementing wholesale conversion to the next software upgrade. Existing network and hardware constraints, combined with the instrument paradigm (that is, the logic and organization of the production instrument in question), may be critical to assessing real-time production performance. This may involve performance of transparent remote (for example, via PCAnywhere) or actual on-site testing of the physical production environment as a final step. The type and extent of external system testing of the instrument functionality is determined by the particular network and server configuration and will involve coordination with the systems technology staff.

Including thorough viability testing into the “scheduled time to next upgrade” is critical to the seamless functionality of your production instrument. Also, it is typically desirable to minimize upgrades (that is, a few times yearly), allowing the maximum amount of time on the front end to assess upgrades and instrument integrity in a baselined test environment before installing in the production environment.

Diagnosis and Response

Optimization testing during development and modification of a Blaise instrument will often reveal pressing issues, or specification oversights or nonconformities that must be resolved to maintain

minimum required functionality. Optimization testing, both core and ad hoc, are typically also one reliable source of diagnosing inefficiencies or inconsistencies that do not threaten performance but should be corrected to improve or maximize functionality. We have implemented both a tracking log and a set of core testing procedures, as well as a structured yet flexible system of testing within the appropriate environment using appropriate data and measuring tools, for speedy resolution of both straightforward and thorny issues.

Summary

Testing is critical to the functionality of a production TFU Blaise instrument. Core testing procedures based on rigorously maintained specifications, along with coordinated versioning and archiving, are critical for baselining instrument functionality. Additionally, procedures must also be developed to accurately track, diagnose, test, and resolve real-time issues and spontaneous problems in a timely manner. Regular implementation of both core and problem-oriented testing results in better conformity to specifications. It also allows for preemptive treatment of problems before they reach production and also for enhancement of instrument functionality through diagnosis of inefficiencies.

Acknowledgements

Amanda Foster Sardenberg is a computer specialist who also performs processing maintenance and production instrument management for the ACS TFU project.

John W. Gloster is a survey statistician who also maintains and updates specifications and procedures for production instrument management on the ACS TFU project.

A Practical Application of Audit Trails

Rob Bumpstead, Office for National Statistics

Introduction

As Computer Assisted Interviewing (CAI) has become the industry standard in the collection of official survey statistics, interest in mechanisms for monitoring the behaviour of those interacting with these systems has grown. Initial attention was focused on monitoring interviewers as they administered CAI questionnaires. More recently, this interest has extended to studying respondent behaviour in various modes of Computer Assisted Self-Interviewing (CASI).

This paper describes the initial stages of a small-scale project, carried out by the Social Survey Division of the Office for National Statistics (ONS), to evaluate the utility to the organisation of the monitoring method known as an audit trail, which has become available in Blaise in recent years. An ONS survey of children and adolescents in the care of local government authorities, known as the survey of Looked After Children (LAC) was chosen as the vehicle for this trial. The Blaise instrument¹ for the survey was designed to collect information about the mental health of children and included a substantial computer assisted self-completion section relating to sensitive or illegal behaviour. The main interest of the trial was to monitor respondent behaviour in the Audio Computer Assisted Self-Interview (A-CASI) section of the interview.

This paper may be of particular interest to those considering use of the audit trail tool, and its associated methodology, for the first time. The emphasis of this paper is upon the practicalities of using an audit trail, and an evaluation of the insights that may be gained by its use. While the work of the project is still on-going, and results incorporating final data are awaited, this paper seeks to explore some of the issues surrounding the capabilities, benefits and difficulties of using audit trails and audit trail data from Blaise.

Capabilities of the Blaise audit trail

The Blaise audit trail provides a record of the values of fields and the movements between fields. As the user enters a field, the value of that field and the time are recorded. When the user leaves the field another record of the value and time is made. In addition, the audit trail can also record use of certain functions, for example, help, save or the use of a searchtag. In this way a detailed, timed record of item response and navigation through the instrument is collected.

There are a number of potential uses that this information may be put to. Perhaps the principal use is methodological; to study navigation through an instrument, use of function keys, speed of data entry, and other aspects of respondent or interviewer interaction with a CAI instrument. What the Blaise audit trail cannot do is record *everything* the respondent or interviewer enters. So if, for example, a respondent begins a question response then deletes their initial answer and enters a different response and then leaves the field, only the final response will be recorded by the audit trail. The audit trail does not produce what is commonly referred to as a 'key-stroke file', that is, a file which details every key-stroke action in the CAI interaction. The limitations of the Blaise audit trail approach are considered later in this paper.

Other potential uses of the audit trail are more prosaic. An audit trail may be used to help debug problems in the Blaise code by tracking the flow of a very complex instrument. The audit trail also offers a means of

¹ The instrument was written in Blaise for Windows 4.2

recovery when data has been corrupted or lost: using the audit trail it may be possible to recreate such cases. Utilisation of audit trails for these purposes are outside the scope of this paper.

Using audit trails on CASI and A-CASI instruments

The 2001 survey of mental health of children in state care was the latest in a decade's work by Social Survey Division in the field of mental health survey research. The questionnaire for this survey is based upon that of previous survey of mental health among children living in private households, conducted in 1999. This 1999 survey used a combination of CAPI and CASI when interviewing young people. The interview included a number of problematic topics such as troublesome and violent behaviour, drug use and sexual activity. The ability of self-interview methods to increase reporting of sensitive, undesirable or criminal behaviour has been well established in the survey literature and therefore, a range of questions relating to such behaviour were included in the CASI section of the interview. (For example, Bradburn 1991; de Leeuw and Nicholls 1996; Tourangeau 1996)

The 2001 survey presented additional problems for the researcher. Much of the subject matter was similar to the previous survey and so a self-interview section was highly desirable. However, the children in state care were much more likely than their counterparts in private households to have problems with literacy and other learning difficulties. In addition, the prevalence of behavioural problems and problems with concentration among the sample population was also known to be very high. These factors could militate against the successful implementation of CASI, and so it was decided to use A-CASI. It was hoped that the audio option would enable a greater number of young people to complete the section, and that the added stimuli would help to engage the respondents. In addition, the heightened sense of privacy that A-CASI is sometimes theorised to provide, may further encourage disclosure of sensitive behaviour. Respondents certainly appear to prefer this mode of self-completion when compared with CASI (O'Reilly 1994).

Audit trail data could potentially play a useful role in assessing the success of these methods. The research team gathered as much information as possible about the impact of A-CASI; they questioned both the interviewers and respondents about their observations and experiences of this section of the interview. (Gatward 2001). However, an audit trail could provide much more information about the child's navigation through the instrument, their ability to use the keyboard and other special key functions available to them.

Perhaps most importantly the audit trail would be a means to ascertain whether, as well as providing answers, the children in the study were actually *listening* to the questions. The Blaise multi-media function provides the capability to complement or replace visually presented questions and answers with audio ones that may be heard through personal headphones. However, the respondent may interrupt the audio recording whenever they choose, enter an answer and move on to the next field. By comparing the audited time spent by the respondent in each field with the length of the sound file for each question, an assessment of the extent to which questions were, or were not, listened to could be made.

Implementing the audit trail

The process of implementing an audit trail is quite straightforward. First, the appropriate audit trail dynamic link library (DLL) must be selected or developed. The audit DLL is an external file which enables the audit trail to be written. The Blaise system provides two: audit DLL and auditkey DLL. The former writes all audit information into one file, the latter into separate files for each form according that form's primary key. The auditkey DLL thus provides distinct and identifiable files for each session in each form and was the DLL chosen for the trial. Although not all the information produced by this DLL may be needed it provided a starting point. Customising the DLL to the specific requirements of the survey could be considered at a later stage.

This DLL was installed on interviewer laptops along with the survey instrument. The audit trail was invoked by adjusting the audit trail section of the mode library editor. The 'make audit trail' box was ticked and the audit trail DLL name and path specified. The instrument was then compiled using the new mode library.

The audit output information from a DEP session is stored in an .adt text file, its content and appearance determined by the audit DLL. For each .adt file the standard Blaise DLL produces a chronological series of records based on events. Each record may contain up to five different fields. The first two fields are common to all records: a date timestamp field recording the date and time that a particular event occurred, and, an action field containing a record of any form or field event in the Blaise database due to menu selection or certain keystrokes. Most commonly, the actions recorded here will be 'leave field' or 'enter field' together with a specified field name, but a range of other actions can also be recorded. In addition, metafile information may be stored in this field, for example, the metafile name. Any other record details, including values, remarks and field status, are contained in up to three further fields in the .adt output. In the event of a 'leave field' action, details of the 'cause' will always be recorded: for example moving on to the next field by answering a question or navigating through the form by arrow or function keys.

In the trial, the initial installation and testing of the audit trail DLL on office computers and laptops was successful. However the audit trail did not appear to function properly when used on interviewers' laptops. The data transmitted back from the field included an .adt file for each interview, recording the timing of the beginning and the end of each session, but no information about interactions of the session itself.

Initially, attention centred on difficulties with memory as a possible cause of the problem and the interaction between the data entry programme, the questionnaire and the audit DLL. The problem might relate to the capabilities of the laptop computers used by interviewers and the large size of the questionnaire, which contained numerous sound files. This highlights one of the potential difficulties of using audit trails. A procedure which records every movement through a questionnaire from one field to another, and records values together with dates and times for each movement, can quickly generate an extremely large amount of information which can either affect performance or overwhelm the disk space available. However, reports from interviewers in the field seemed to indicate that laptop performance had not been noticeably affected.

Eventually the problem was diagnosed, not as a memory issue, but relating to the auditkey.dll source code and its interactions with the ONS case management system CaseBook. In summary, while the .adt file containing information about the session interaction was generated, it was written to an unexpected location on the interviewer laptop. This meant the .adt file was not transmitted back to the office along with the interview data.

While these problems were being overcome, the trial continued to the next stage of constructing processing systems and building an analytic model.

Processing the audit trail data

The detail and sheer volume of data produced by the audit trail presents the immediate problem of how to effectively conduct analysis. In the first instance, it is of course possible to examine each audit output by hand. Where the number of cases is relatively small, or where interest is limited to a few specific instances in the instrument, this may be a viable approach. In this example, the A-CASI pilot data was expected to consist of small number of cases (n=50). On the other hand, while the general area of interest was confined to audit output relating to the CASI sections of the questionnaire; this in itself consisted of over 60 questions and produced hundreds of audit records for each form. When targeting a particular question or question type, for example the handful of instances where an open text answer was required, the 'by hand' technique may be the most effective way of evaluating the audit data. However, in most instances the need to summarise the data in some form was plain.

Processing the audit trail file can be done via Manipula. Two standard Blaise programs are provided for this purpose one of which, auditsummary.man, was useful in this context. It summarises the contents of the data by keeping only the last value of any field and deleting all other lines. Further basic improvisations can readily be made to the Manipula programs to streamline the presentation of data and select particular areas of interest. Another tool for summarising data was kindly made available to us by colleagues at the Bureau of the Census in the United States, who had developed an audit trail report generator utility². The purpose of this utility was to convert the contents of raw .adt output files to a simplified format, summarising the time a user spent within given sections of the questionnaire. A combination of all these methods was employed in preparing an analytic model for studying the LAC data.

Analysing the audit trail data

The first element of the model was concerned with timestamp data. The Blaise Audio-CASI method permits the respondent to interrupt an audio file as it is being played. The respondent may therefore listen only to some of the question, or if they listen to the full question, only some of the answer categories. This timestamp audit data can be used to reveal the extent to which interrupted listening has occurred. In developing a strategy for this analysis previous work in this field by Caspar and Couper was particularly useful (Caspar and Couper 1997).

A series of questions were selected for analysis from each of the following sequential sections of the questionnaire: moods and feelings, troublesome behaviour, alcohol and illegal drug use, sexual activity and exclusion from school. Initially, only those questions asked of all respondents were included to simplify the analysis. In the interview the question and answer options were played via headphones, while only the answer options were displayed on-screen. For each field the length of the sound file for the question and the answer categories was calculated. This could be compared with the actual time spent in each field by the respondent, and used to calculate two 'listening scores'.

The first listening score relates to the question only, and reflects the proportion of the question sound file the child had listened to. A score of one would indicate that the child had listened to the full question, a score below one, that the question sound file had been interrupted. The second listening score is a combination of both the question and answer categories, and can be expressed as a ratio. For example, if a child spent 20 seconds in a field where the combined time of question and answer categories was 10 seconds, the listening ratio would be 2. The differences between 'recorded time' and 'actual time' could then be used to produce an aggregate ratio for each section, and overall patterns of behaviour studied.

The implications of a low listening score for respondent understanding vary according to the wording of particular questions and answer options. A simple question may be adequately comprehended before the answer options have been heard or seen, or even before the full question text has been heard. On the other hand, uninterrupted listening may be essential for questions which end with important qualifying phrases. The respondent may need more time to consider questions where essential elements appear in the response categories.

The audit data can be used at the individual question level to explore these effects. At the aggregate level, listening scores can be assessed and compared across sections. We might expect scores to fall in general as the user progresses through the questionnaire and becomes more familiar with the format, and the audit data could be used to examine this theory.

A second element of the analysis was concerned with use of specific keys. In particular, a repeat function was provided to respondents so that the recording of a question could be replayed. Monitoring the use of this function would enable an assessment of how useful respondents found it to be, but could also highlight any

² We gratefully acknowledge the help of Ellen Soper from the US Bureau of the Census

problems in understanding particularly complex or difficult questions, or where the sound quality of a question had been insufficient.

Use of the keyboard outside the range of specified keys could also be examined. For most questions, respondents were instructed to use only the 'enter' and 'repeat' keys (labelled blue and white) together with the numeric keypad. Use of other keys could indicate difficulties in understanding the instructions, using the keyboard, or problems with concentration.

The final element of the analysis focussed on the children's use of the limited number of open text boxes that the self-completion section of the instrument contained. Because the audit trail is not a keystroke file the extent to which behaviour in these boxes can be monitored is limited. Only the value of the field before and after entry is recorded. Indeed, it is not possible to capture 'within-field' deliberation of this type in any question response, though the number and length of 'revisits' to a question field could be studied.

However, the audit data does permit consideration of another aspect of respondent reaction. Open text questions only arise given certain responses to previous questions. Using the navigational data contained in the audit records it is possible to see whether respondents have 'backed-up' after reaching such a question and altered a previous answer to change their path through the questionnaire.

Conclusions and further work

In conclusion, the Blaise audit trail can provide useful data about respondent use of CAI instruments which it would not otherwise be possible to capture. In relation to A-CASI, the provision of 'listening scores' of the type described could be particularly valuable. The processing and analysis of audit data is not straightforward. However, it was possible to formulate a strategy for dealing with audit data which was not unduly cumbersome or time consuming, given the proviso of a relatively small survey dataset. The lack of a key-stroke capture facility in the Blaise audit trail did limit some of the options for analysis, but was not a serious impediment to the overall methodology proposed here.

The problems in fully operationalising the audit trail in the field were, in the short term, a more serious concern. Having been overcome, the next stage of the program could be to further develop and automate a processing system so that greater quantities of audit data might be efficiently analysed. This could include using other existing audit trail utilities such as WesAudit as well developing in-house systems. Another important step could be to customise the standard DLL used in this trial to tailor audit output. In these ways the utility of the audit tool might be further enhanced: while the application of the Blaise audit trail may in practice be best employed in testing relatively small quantities of data and experimental designs it is nevertheless a valuable tool.

References

Bradburn, N., Frankel, M., Hunt, E., Ingels, and Pergamit, M., (1991) "A Comparison of Computer-Assisted Personal Interviews with Personal Interviews in the National Longitudinal Survey of Labour Market Behaviour —Youth Cohort", *Proceedings of the US Bureau of the Census Annual Research Conference*, Washington, DC: US Bureau of the Census, pp. 389 - 397

Caspar, R., and Couper M., (1997) "Using Keystroke Files to Assess Respondent Difficulties with an Audio-Casi Instrument", *Survey Research Methods - Proceedings of the American Statistical Association*, pp239-244

de Leeuw, E., and Nicholls, W., (1996) "Technological Innovations in Data Collection", *Sociological Research Online*, 1:4 <<http://www.socresonline.org.uk/socresonline/1/4/leeuw.html>>

Gatward, R., (2001) "Audio-CASI with challenging respondents", paper in this volume, 7th International Blaise Users Conference

O'Reilly, J., Hubbard, M., Lessler, J., Biemer, P., and Turner C., (1994), "Audio and Video Computer Assisted Self Interviewing", *Journal of Official Statistics*, 10, 2, pp. 198-214

Tourangeau, R., and Smith, T., (1996), "Asking Sensitive Questions: The Impact of Data Collection Mode, Question Format and Question Context", *Public Opinion Quarterly*, 60:275-304

Automated Testing of Blaise Questionnaires

Jay R. Levinsohn, Research Triangle Institute
Gilbert Rodriguez, Research Triangle Institute

INTRODUCTION

Social science data collection has seen a significant movement toward the use of automated data collection procedures, using increasingly sophisticated tools. The Blaise programming language and data collection system is one example of this kind of data collection tool. While we see increasing numbers of social scientists designing large and complicated data collection instruments under Blaise we have not seen the parallel development of tools to aid in the testing and certification of these instruments. As the specification for a given question grows in size and complexity its implementation into a Blaise program grows into a significant software development project. While there are tools to help in the software development process in Blaise we do not see the tools that support quality assurance and validation.

At the Research Triangle Institute (RTI) the National Household Survey of Drug Abuse project uses the Blaise system to collect computer assisted personnel interview data from almost 70,000 individuals each year. This data collection effort is an annual effort with quarterly data collection waves each year. The questionnaire is large and increases in programming complexity each year. The typical quality assurance (QA) methods of manually testing the instrument for adherence to the designers specifications, accurate program logic and flow, correct data storage and error free transmission of the data is an increasingly time consuming, expensive, slow, and error prone set of procedures. The QA process becomes even more problematic as one repeats it over the inevitable questionnaire revisions; each with tighter timelines that occur as a given questionnaire approaches its production deadlines.

Over the past year RTI has been developing a set of tools to allow the Blaise developer to automatically test some components of the questionnaire implementation. RTI has developed a prototype system that allows the questionnaire developer to create “test scripts” that can be fed into a working Blaise questionnaire implementation, in the actual data collection configuration, as a keystroke stream. The test scripts are developed to exercise the flow, variable calculation, sub-sampling, and all major paths through the questionnaire. As the script stream is fed into Blaise, the program flow is monitored with respect to the correct question sequence (as defined by the script) and the correct exit point from the questionnaire. Testing logs are generated for each script documenting time and date of testing, screen by screen progress through the questionnaire, and the data entered on each screen. The data output from a given script goes into the Blaise database as a simulated questionnaire completion. The Blaise database can then be compared to the script input for validation or as the case at RTI we actually transmit the data, simulating field procedures, through the laptop transmission system and then validate the data received against that expected from the script.

This relatively simple process provides several benefits:

- Complete end to end testing of keystroke input against received data output
- Documentation and repeatability of the testing process
- Validation of specific paths through the instrument, one per test script
- Rapid re-testing as one steps through the final iterations of the development process (regression testing)

This process still leaves significant elements of a questionnaire untested:

- Screen text validation
- Answer text validation
- Sound file validation
- Paths and calculations not covered by the range of test scripts

RTI is continuing to investigate adding new elements to the testing system. We are currently working on a component that would allow “Random Walks” through the questionnaire to test for logic errors, flow errors, and range errors. In addition we are looking at development of a more formal process for creating questionnaire specifications that might facilitate testing screen and answer text.

THE METHODOLOGY

The goal of this testing process is to insure that a given Blaise program faithfully implements the specifications of the questionnaire designer. Figure 1, shown below, depicts the process of designing, developing, testing and then rolling out a questionnaire. The figure shows a process in which a design leads to written specifications that are then prototyped. The prototype is reviewed and tested. The design specification stage may be repeated, multiple times, until the designers, testers, and reviewers feel that the questionnaire is correct and ready for production testing. At this stage a round(s) of coding, test and review is done until all parties feel the instrument is correct. In terms of the work we are doing here being correct implies that the program implements the written specifications. In all cases, if errors or the need for change are found the design documents and written specifications must be changed, followed by another round of testing.

This process if conducted without error should provide a Blaise questionnaire that treats a given sequence of interviewer inputs according to design specifications. The output from a specific set of interviewer inputs should produce a predictable set of data outputs in the Blaise database. Using this idea we have developed software and a set of procedures that tests Blaise questionnaires with the goal of verifying that a specific set of inputs (as codified in a script) produces a predictable set of Blaise outputs. Figure 2 highlights the points at which the system tests. As shown in Figure 2 we are trying to verify that the specification developed and that data produced in the database, given a known set of inputs, are in concert. In order to do this we developed the idea of creating “Test scripts” that are derived from using only the questionnaire specifications. These scripts would then be feed into the Blaise program. If the program is correct we can predict exactly which fields and which data values should be in the Blaise database. This is essentially the model of testing that is done manually when a designer walks through a Blaise program to test the program flow and data validation. The tester would enter data and using their knowledge of the questionnaire specifications, then review and confirm the data entry and routing through the questionnaire.

This testing process is implemented in a collection of software tools that we have named RoboCAI. RoboCAI uses the following tools:

- WinBatch - a batch language interpreter for Windows. Batch files are written using WIL, the Windows Interface Language. A batch file can contain commands to run programs, read and write directly to files, send keystrokes to Windows applications, and perform a number of other tasks.

Using the WinBatch compiler, an executable file can be created from a batch file. WinBatch is a proprietary product developed by Wilson WindowWare, Inc.

- Blaise 4.x – CAI instruments created with different versions of Blaise 4.x have been used with RoboCAI. Currently, Blaise 4.5 is being used.
- A modified audit trail DLL, audit.dll – This DLL, written in Delphi, writes out the name of the current field in an instrument to a text file where it can be read by a WinBatch batch file or executable.

When executed, the RoboCAI program file prompts the user for the name of a text file containing response data corresponding to an interview case (which is called a script file) and then prompts the user for the name of a log file to which diagnostic data is written. The RoboCAI program starts CAI instrument, reads the response data file, and then sends the response data keystrokes to each screen of the CAI instrument. The audit trail DLL writes the name of the current screen name of the CAI instrument to a text file (currentfield.txt) so that the RoboCAI batch file can compare the script file routing to where the CAI instrument reports as its current location. Figure 3 presents a flow diagram of these steps.

The flow and logic of RoboCAI is simple with the one complication of coding to allow for synchronization of the activity between the Blaise audit.dll and the Winbatch code. These two processes, RoboCAI and Blaise, run in parallel under the Windows system and the RoboCAI code must wait for the Blaise code to complete. The RoboCAI code must allow the Blaise audit.dll enough time to open, write into, and close the screenfile.txt file before it can check to see if it is still tracking the flow through the questionnaire correctly.

An additional software module was written, in Visual Basic, that completes the last part of the test sequence. This code compares the extracted text file from Blaise that contains the data stored during the RoboCAI phase to the data input from the script. The comparison process documents the test and highlights any differences between script data and Blaise data.

EXAMPLE

The following sections present a short RoboCAI example. In Tables 1 through 4 we present the information that serves as input into the RoboCAI test process and the outputs. In this example scenario the "respondent" is a 21 year old male from Wyoming who is on active duty in the military. Hence, for this example survey, he will be routed out of the interview very early since he is ineligible to participate in the survey. QUESTID is an identifier for the interview case and is the primary key for the datamodel.

A. Sample Questionnaire

The CAI questionnaire screens that appear for this scenario are shown in Table 1.

Table 1: A Sample CAI Questionnaire
CAI SCREENS FOR QUESTID 2000024

Question Name: startup
INTERVIEWER: SELECT THE LANGUAGE TO BE USED FOR THIS INTERVIEW.
1 ENGLISH

2 SPANISH
3 MULTIMEDIA LANGUAGE

Response: ENGLISH

Question Name: note1

FI: DO NOT READ ALOUD UNLESS RESPONDENT QUESTIONS THE BURDEN (OR TIME) ASSOCIATED WITH THIS INTERVIEW.

NOTICE: Public reporting burden for this collection of information is estimated to average 60 minutes per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information

PRESS [ENTER] TO CONTINUE.

Response: [Return] to continue

Question Name: remindfi

INTERVIEWER: IF YOU HAVE NOT READ THE "INTRO TO CAI" IN YOUR SHOWCARD BOOKLET ALOUD TO THIS RESPONDENT, DO SO NOW. WHEN RESPONDENT IS FULLY INFORMED, CONTINUE WITH THE INTERVIEW.

PRESS [ENTER] TO CONTINUE.

Response: [Return] to continue

Question Name: age1

What is your date of birth?

ENTER MM-DD-YYYY.

Response: 7-24-1980

Question Name: confirm

That would make you 21 years old. Is this correct?

- 1 YES
- 2 NO

Response: YES

Question Name: FIPE1

INTERVIEWER: WERE 2 PERSONS SELECTED FOR AN INTERVIEW AT THIS SDU?

- 1 YES
- 2 NO

Response: NO

Question Name: FIPE4

INTERVIEWER: IN WHAT STATE IS THIS SAMPLE DWELLING UNIT (SDU) LOCATED?

- | | |
|---|-------------------|
| 1 ALABAMA | 27 MONTANA |
| 2 ALASKA | 28 NEBRASKA |
| 3 ARIZONA | 29 NEVADA |
| 4 ARKANSAS | 30 NEW HAMPSHIRE |
| 5 CALIFORNIA | 31 NEW JERSEY |
| 6 COLORADO | 32 NEW MEXICO |
| 7 CONNECTICUT | 33 NEW YORK |
| 8 DELAWARE | 34 NORTH CAROLINA |
| 9 THE DISTRICT OF COLUMBIA (WASHINGTON, DC) | 35 NORTH DAKOTA |
| 10 FLORIDA | 36 OHIO |
| 11 GEORGIA | 37 OKLAHOMA |

12 HAWAII
13 IDAHO
14 ILLINOIS
15 INDIANA
16 IOWA
17 KANSAS
18 KENTUCKY
19 LOUISIANA
20 MAINE
21 MARYLAND
22 MASSACHUSETTS
23 MICHIGAN
24 MINNESOTA
25 MISSISSIPPI
26 MISSOURI

38 OREGON
39 PENNSYLVANIA
40 RHODE ISLAND
41 SOUTH CAROLINA
42 SOUTH DAKOTA
43 TENNESSEE
44 TEXAS
45 UTAH
46 VERMONT
47 VIRGINIA
48 WASHINGTON
49 WEST VIRGINIA
50 WISCONSIN
51 WYOMING

Response: WYOMING

Question Name: FIPE5

INTERVIEWER: THE STATE YOU ENTERED IS:

WYOMING

IS THIS CORRECT?

- 1 YES
- 2 NO

Response: YES

Question Name: QD01

The first few questions are for statistical purposes only, to help us analyze the results of the study.

INTERVIEWER: RECORD RESPONDENT'S GENDER

- 5 MALE
- 9 FEMALE

Response: FEMALE

Question Name: QD03

Are you of Hispanic, Latino or Spanish origin or descent?

- 1 YES
- 2 NO

Response: YES

Question Name: QD04

HAND R SHOWCARD 1. Which of these Hispanic, Latino, or Spanish groups best describes you? Just give me the number or numbers from the card.

TO SELECT MORE THAN ONE CATEGORY, PRESS THE SPACE BAR BETWEEN EACH CATEGORY YOU SELECT.

- 1 MEXICAN / MEXICAN AMERICAN / MEXICANO / CHICANO
- 2 PUERTO RICAN
- 3 CENTRAL OR SOUTH AMERICAN
- 4 CUBAN / CUBAN AMERICAN
- 5 OTHER (SPECIFY)

Response: CUBAN / CUBAN AMERICAN

Question Name: QD05

HAND R SHOWCARD 2. Which of these groups describes you? Just give me a number or numbers from the card.

TO SELECT MORE THAN ONE CATEGORY, PRESS THE SPACE BAR BETWEEN EACH CATEGORY YOU SELECT.

- 1 WHITE
- 2 BLACK/AFRICAN AMERICAN
- 3 AMERICAN INDIAN OR ALASKA NATIVE (AMERICAN INDIAN
INCLUDES NORTH AMERICAN, CENTRAL AMERICAN, AND SOUTH AMERICAN INDIANS)
- 4 NATIVE HAWAIIAN
- 5 OTHER PACIFIC ISLANDER
- 6 ASIAN (FOR EXAMPLE: ASIAN INDIAN, CHINESE, FILIPINO,
JAPANESE, KOREAN, AND VIETNAMESE)
- 7 OTHER (SPECIFY)

Response: WHITE

Question Name: QD07

Are you now married, widowed, divorced or separated, or have you never married?

INTERVIEWER NOTE:

If the respondent is divorced but currently remarried, code as married.
By 'divorce' we mean a legal cancellation or annulment of a marriage.
By 'separated' we mean legally or informally separating due to marital discord.

- 1 MARRIED
- 2 WIDOWED
- 3 DIVORCED OR SEPARATED
- 4 HAVE NEVER MARRIED

Response: MARRIED

Question Name: QD08

How many times have you been married?

Response: 2

Question Name: QD09

Have you ever been in the United States' armed forces?

- 1 YES
- 2 NO

Response: YES

Question Name: QD10

Are you currently on active duty in the armed forces, in a reserves component, or now separated or retired from either reserves or active duty?

- 1 ON ACTIVE DUTY IN THE ARMED FORCES
- 2 IN A RESERVES COMPONENT
- 3 NOW SEPARATED OR RETIRED FROM EITHER RESERVES OR
ACTIVE DUTY

Response: ON ACTIVE DUTY IN THE ARMED FORCES

Question Name: MILTERM1

I need to verify what I just entered into the computer. You said you are currently on active duty in the armed forces. Is that correct?

- 1 YES
- 2 NO

Response: YES

Question Name: MILTERM2

People who are currently on active duty in the armed forces are not eligible to be interviewed in this study. I appreciate you taking the time to speak with me. Thank you.

PRESS [ENTER] TO CONTINUE.

Response: [Return] to continue

Question Name: fiexit

End of interview reached.

PRESS 1 TO EXIT.

Response: 1

B. SAMPLE SCRIPT

Table 2 displays the corresponding RoboCAI script. When the RoboCAI program file is executed, it starts the CAI instrument, reads in this script and sends the appropriate keystrokes to each screen of the CAI instrument. The script file syntax allows for a totally blank line which is ignored, a line starting with a “;” which indicates a comment line and a line containing three fields each separated by a tab character. The three fields are: screen name, data value, and an optional comment (denoted by starting with a “;”).

Table 2: A Sample RoboCAI Script

```
QuestID      2000024
; 10/24/2000  short one only military
;
; call startup and intro demographics
;
startup 1
note1
remindfi
age1 07-24-1980
confirm 1
fipe1 2
fipe4 51
fipe5 1
;
QD01 9      ;sex of R 9=female
QD03 1      ;Hispanic? 1=Yes, 2=No
QD04 4      ;kind of Hispanic (1-4, 5=other)
QD05 1
QD07 1
QD08 2
QD09 1
QD10 1
Milterm1     1
Milterm2
Fiexit 1
```

C. SAMPLE LOGS AND COMPARISON OUTPUT

The text displayed in Table 3 is the output log file generated by the RoboCAI program file. The log file contains the data that were used as input and lists any error or information messages from the test run. RoboCAI compares the screen names appearing in the script file to the actual screens encountered and writes out error messages to this file if any discrepancies occur. It also reports the data supplied to Blaise from the script.

Table 3 RoboCai Logs			
RoboCAI Script Testing			
Mon 5-07-2001 11:32:31 AM			
	questID	2000024	
	Blaise Screen	Script Screen	Script Data
	-----	-----	-----
1	STARTUP	STARTUP	1
2	NOTE1	NOTE1	
3	REMINDFI	REMINDFI	
4	AGE1	AGE1	07-24-1980
5	CONFIRM	CONFIRM	1
6	FIPE1	FIPE1	2
7	FIPE4	FIPE4	51
8	FIPE5	FIPE5	1
9	QD01	QD01	9 ;sex of R 9=female
10	QD03	QD03	1 ;Hispanic? 1=Yes, 2=No
11	QD04	QD04	4 ;kind of Hispanic (1-4, 5=other)
12	QD05	QD05	1
13	QD07	QD07	1
14	QD08	QD08	2
15	QD09	QD09	1
16	QD10	QD10	1
17	MILTERM1	MILTERM1	1
18	MILTERM2	MILTERM2	
19	FIEXIT	FIEXIT	1
=====			
End of Script			
Mon 5-07-2001 11:33:00 AM			

As mentioned in a previous section, an additional Visual Basic program was written that compares the extracted text file from Blaise to the data input from the script. Any differences found between script data and Blaise data are highlighted in a log. Table 4 displays the contents of the output log file from this program for this example. In Table 4 one difference is highlighted with “****”. This difference stems from the difference in the date storage, Blaise converts input dates for storage as Day-Month-Year but the data was input as month-day-year.

Table 4: Comparison of Script Data and Blaise Data

05-07-2001 5-7-2001 11:49:20 AM c:\roboCai 2001\24out.txt

Seq #	Field Name	Script	Data File
2	STARTUP	{1}	[1]
5 ****	AGE1	{07-24-1980}	[24071980]
6	CONFIRM	{1}	[1]
7	FIPE1	{2}	[2]
8	FIPE4	{51}	[51]
9	FIPE5	{1}	[1]
10	QD01	{9}	[9]
11	QD03	{1}	[1]
12	QD04	{4}	[4]
13	QD05	{1}	[1]
14	QD07	{1}	[1]
15	QD08	{2}	[2]
16	QD09	{1}	[1]
17	QD10	{1}	[1]
18	MILTERM1	{1}	[1]
20	FIEXIT	{1}	[1]

SUMMARY, CONCLUSIONS, FUTURE WORK

For our purposes, RoboCAI has proved to be effective. RoboCAI becomes very useful as the size and complexity of an instrument increase. While not intended as a means of checking the screen and answer text of an instrument or sound file validation, it does provide a means for checking the correctness of routing and the resulting data for an instrument. In particular, we have used it to perform regression testing on new versions of CAI instruments prior to deploying them for field use and for acceptance testing when upgrading to newer versions of Blaise.

Some possible topics for future work include,

- Work on automated script generation
- Develop a random walk application
- Investigate the use of Blaise API objects to extend testing capability of RoboCai
- Allow specification of date and time of test environment

Regarding the first item, a Visual Basic application has been developed that uses the Blaise API objects to write out RoboCAI scripts using Blaise data files as input. Thus, a user can test an interview case and if the results are acceptable, a corresponding RoboCAI script can be generated. The script can then be used as part of a test library for regression testing and for instrument certification.

Figure 2 -- Test Goals

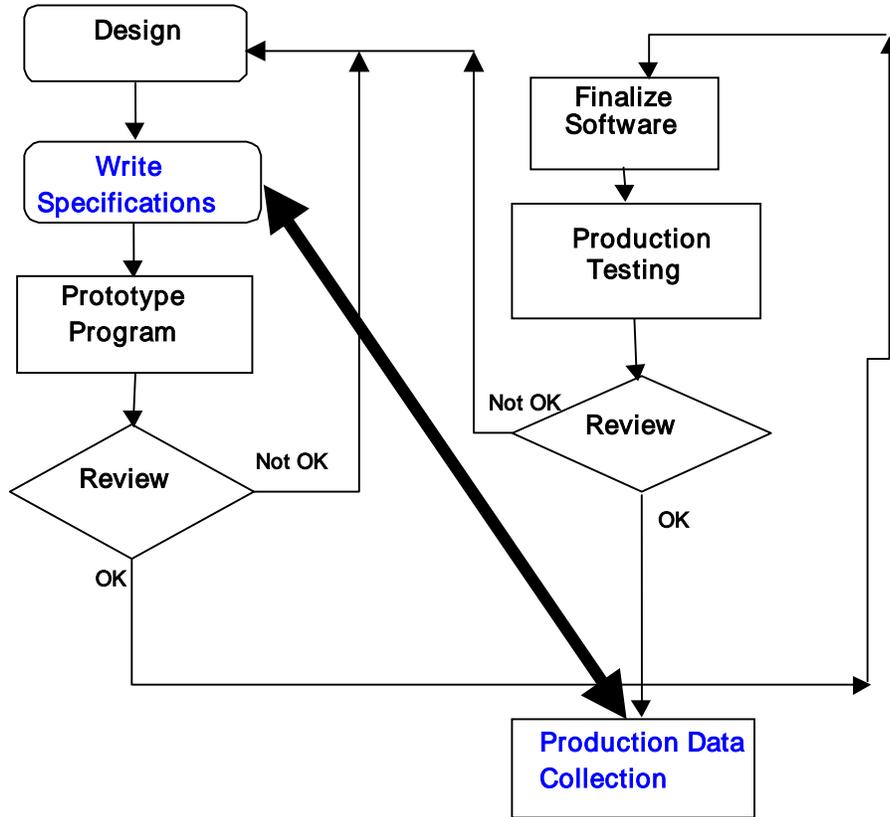
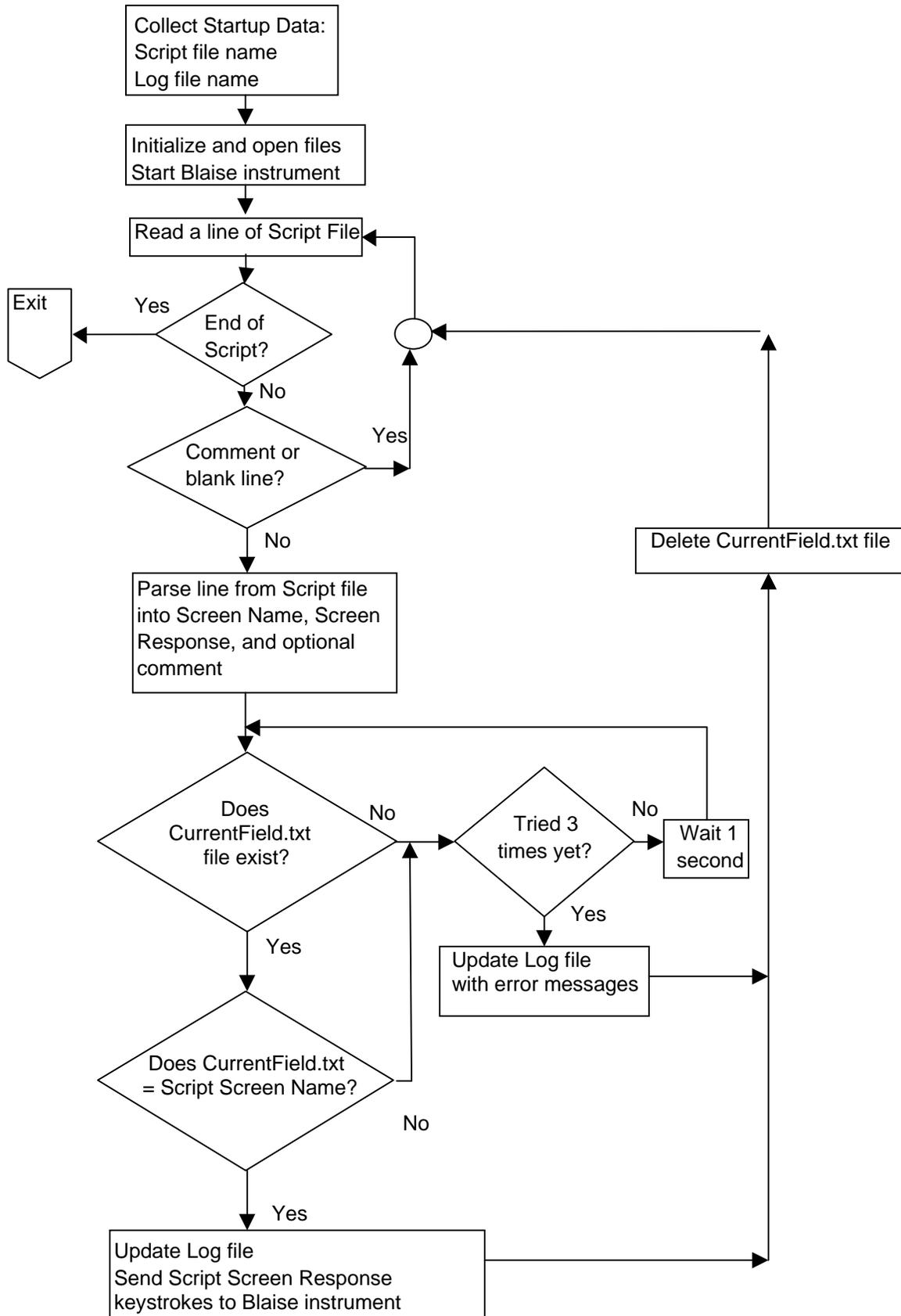


Figure 3 -- RoboCAI



Reporting on Item Times and Keystrokes from Blaise Audit Trails

Sue Ellen Hansen, University of Michigan
Theresa Marvin, University of Michigan

1. 0 Introduction

Research has shown that trace and keystroke files are useful both for pretesting survey instruments and for evaluating interviewer performance (Couper, Hansen, and Sadosky, 1997; Couper and Schlegel, 1998). In Blaise, these files are called audit trails. In CAI systems that provide trace or audit trail files, they generally provide (1) the times that the interviewer entered and exited input fields or items, and (2) actions invoked at specific items, such as backing up, requesting online help, and recording or changing comments or remarks. Some systems, as does Blaise, also provide the option of recording every keystroke the interviewer presses during an interviewing session. Audit trail files with keystrokes are particularly useful in troubleshooting problems interviewers experience using computer assisted instruments and software.

In an analysis of trace files¹ from the 1997 National Health Interview Survey (NHIS) computer assisted personal interview (CAPI) instrument, Couper and Schlegel (1998) were able to identify a number of problematic survey items or screens, that is, screens with high levels of online help access or changed answers following back ups. These findings led to improved design of some of the NHIS screens. In an earlier study, Couper *et al.* (1997) examined keystroke files² from the 1993 CAPI Health Dynamics of the Oldest Old (AHEAD) CAPI study. They demonstrated the usefulness of keystroke files for exploring the effectiveness of interviewer training and for evaluating interviewer performance in the use of computer assisted interviewing (CAI) software and survey instruments (e.g., the extent to which interviewers access online help, successfully back up and change answers, press keys in error or make mistakes in the use of function keys).

This paper discusses the Institute for Social Research's (ISR's) current use of Blaise audit trails. It describes how to set up Blaise to capture audit trails, and then provides an overview of the features of a program ISR uses to read the Blaise audit trails and report on item, block, and interview times. Examples of actual audit trail files and data provided by the reporting program are shown in Section 2, demonstrating how one can use audit trails to evaluate survey instruments and interviewer performance. The final section of the paper describes planned enhancements to the audit trail reporting program, and presents an example of additional information provided by a current beta version that reports on an expanded set of actions.

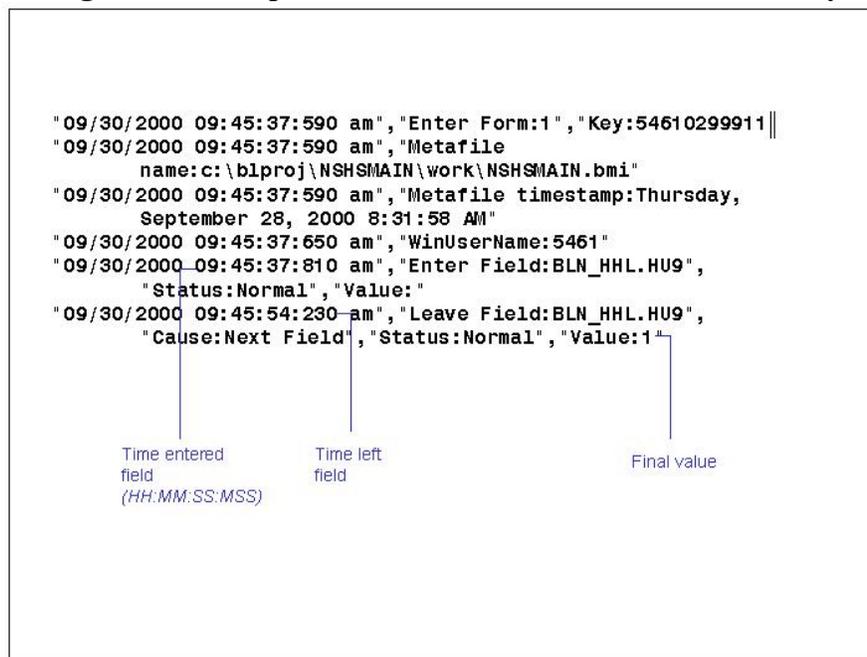
¹ The 1997 CAPI NHIS instrument was programmed in CASES. CASES trace files capture item level times, some commands such as backups and jumps, as well as the final stored response or value for each item accessed. Trace files also can be used to play back or review partial or complete interviews.

² The 1993 AHEAD CAPI instrument was programmed in SurveyCraft. SurveyCraft keystroke files capture item times, every interviewer keystroke, and can be used to play back interviews.

1.1 Blaise audit trails

Statistics Netherlands provides two Dynamic Linked Library (DLL) files that permit users to record audit trails for all Blaise cases, one that gives basic time data, and the other that records time data and keystrokes. ISR has edited both of these DLLs so that they provide time data to the millisecond (the Blaise default files provide this information to the hundredth of a second). The default DLL that captures keystrokes records numeric codes for all keystrokes, for which Statistics Netherlands has provided a translation table. ISR has used this table to modify the keystroke DLL so that it records short names for each keystroke (e.g., ENTR, BACK, etc.), making the audit trails more readable. It also has added the time when the first keystroke at an item is pressed³. The modified DLL files, *AuditkeyISR.dll* (time data only) and *KeyStrokesISR.dll* (time data with keystrokes) produce audit trails with the extensions *.ADT* and *.ADK*, respectively. As part of its sample management system, ISR captures interview lengths from Blaise audit trails⁴, and thus requires all Blaise projects to create an audit trail for every Blaise case accessed, either a basic audit trail or an audit trail with keystrokes. Figures 1 and 2 show sample segments from each of these types of files.

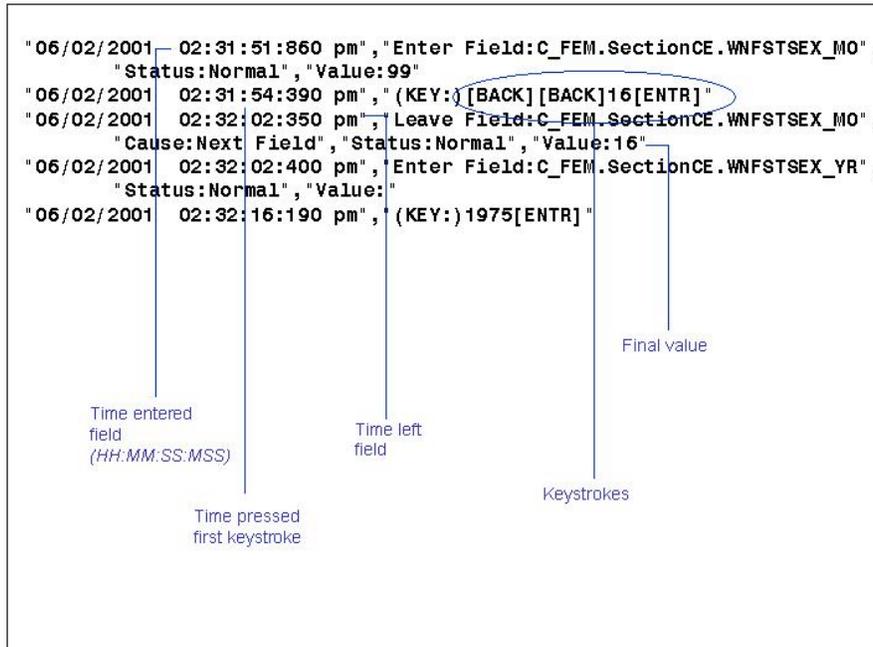
Figure 1. Example from Audit Trail with Time Data Only



³This allows for the calculation of the time between entering the field and the first keystroke, which could serve as a measure of response latency (Bassili, 1996).

⁴Although it is expected that survey instruments are programmed with essential time stamps, such as start of interview and end of interview, and perhaps key section times, ISR has found that audit trails are a more reliable source of survey timing data (Marvin *et al.*, 2000).

Figure 2. Example from Audit Trail with Time Data and Keystrokes



1.2 Setting up a Blaise project to create audit trail files

There are two things that ISR does to ensure that audit trails are captured for each case for a project. First, programmers modify default mode library settings to point to the correct DLL for the project. To confirm that the project collects audit trails with appropriate information, they check the mode library *Style Options*. They make sure that the *Audit Trail* box in the Options window is checked and that the file name for the audit trail format is *AuditkeyISR.dll* or *KeyStrokesISR.dll*. They also make sure that the appropriate directory *path* is specified, and that the selected DLL is in that directory (see Figure 3).

Second, programmers specify in the Blaise data model the variable that will be used to identify each Blaise interview or partial interview, which also uniquely identifies each audit trail file saved⁵. This is the “primary key” for the case, which generally immediately follows the SETTINGS in the data model. Figure 4 provides sample Blaise code that establishes the primary key used to identify each audit trail file.

⁵ ISR sample management systems expect separate .BDB and .ADT (or .ADK, if audit trail has keystrokes) files for each case accessed.

Figure 3. Selecting the Appropriate Audit Trail DLL in the Mode Library

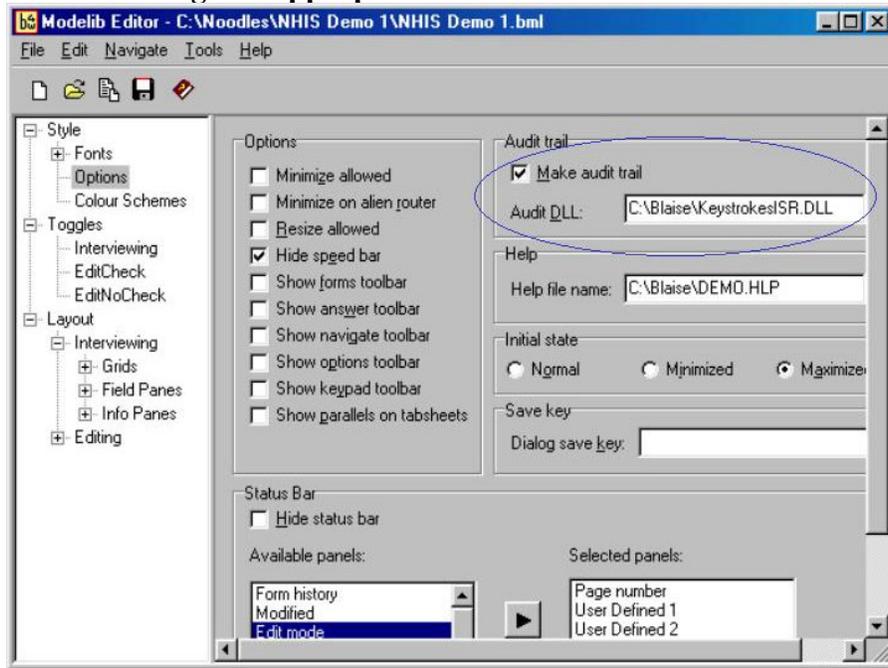


Figure 4. Establishing a Primary Key in the Blaise Datamodel

```

DATAMODEL PrimKey "Primary Key Sample"

SETTINGS
  ATTRIBUTES = DONTKNOW, REFUSAL

PRIMARY
  SampleID
  
```

1.3 Processing audit trail files (ADT_Report.exe)

The program *ADT_Report.exe* was created in Delphi to transform the information in ADT and ADK files into a more useable format. *ADT_Report.exe* is designed either to run in batch mode whenever new Blaise data files are merged into a project master data file or to run interactively with a Graphical User Interface (GUI). When run in batch mode, the program creates a file that provides interview lengths (form total times).

Figure 5 shows the GUI for the interactive version of the program. It requires the following information:

- *Summary levels*, the levels at which times are reported:

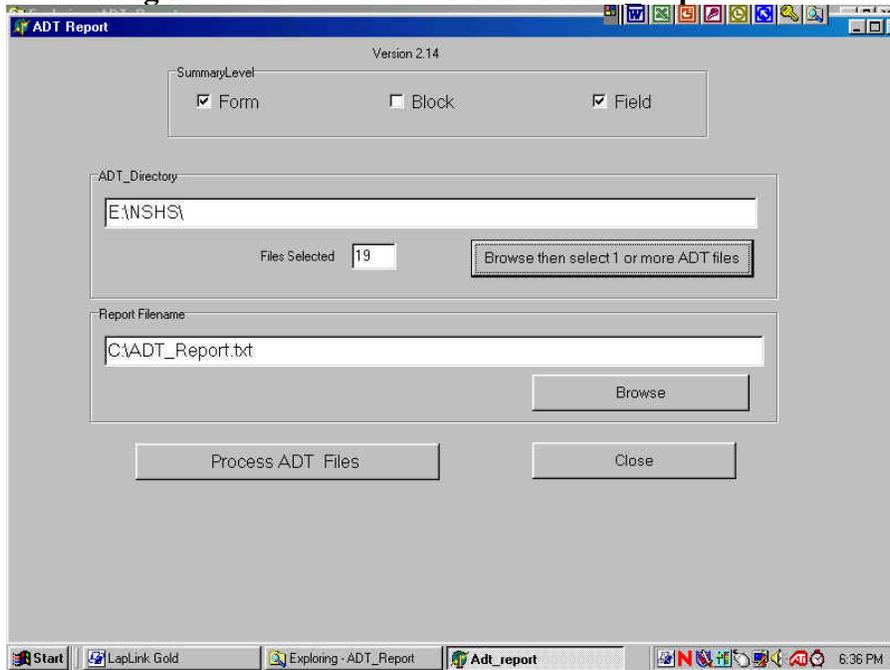
Form	Interview
Block	Block, for instrument sections programmed as blocks (for example, block <i>Section A</i>)
Field	Question (for example, field <i>MarStat</i>)

By default, ADT_Report.exe reports times at the interview and question levels. Users who wish to report block times click on the Block box under Summary Level. To deselect Form, Block, or Field, they would click on the appropriate checked box.

- *The location and name(s) of one or more ADT or ADK files.* Users click on the *Browse then select 1 or more files* button, select the appropriate drive and folder, and select one or more files. By default the program looks in the folder from which it is running (*E:\NSHS* in this example). After completing the selection, the program displays the selected folder and the total number of files selected (*19* in this example)
- *The location and name of the report file.* The *Report Filename* input box displays the location and name of the time report. By default this is *C:\ADT_Report.txt*. To change this, users click on the *Browse* button below this box, select the appropriate drive and path, and enter a new file name. In addition to changing the location of the file, users can also change the base name to reflect the project (e.g. *C:\ADT_Report_ProjectName.txt*). The filename entered here will be used as the base name for any future reports generated by *ADT_Report.exe* (see Section 3).

Finally, to generate the report, users click on the *Process ADT Files* button, and then click on the *Close* button to exit *ADT_Report.exe*.

Figure 5. Interactive Version of ADT Report.exe



1.3 Audit trail time reports

The report named in the *Report Filename* box reports at the level(s) specified in *Summary Level*. The program by default reports item and interview times. Interviewers or cases are identified as forms in Blaise. A report is an ASCII text file. Each line in the file is a record with at least the following fields, exclamation point delimited:

- CASEID Unique project-specific case ID (Blaise primary key or file name)
- FLD_SEQ Sequence in which the field appears in the interview
- FORM_VST Number of current visit to a form (for example, “2” if second visit across current and all prior sessions of the interview)
- FLD_VST Number of current visit to a field (for example, “3” if third visit across current and all prior sessions)
- FLD_NAME Name of field (e.g., MarStat)
- FLD(H:M:S:Mss) Elapsed time between entering field and leaving field (hours-minutes-seconds-milliseconds)
- FLD(SSSS.mSS) Elapsed time between entering field and leaving field (seconds and milliseconds)
- BTW_FLD(H:M:S:Mss) Elapsed time between leaving field and entering next field (hours-minutes-seconds-milliseconds)
- BTW_FLD(SSSS.mSS) Elapsed time between leaving field and entering next field (seconds and milliseconds)
- ADJ_FLD(H:M:S:Mss) Sum of field time and following between time (hours-minutes-seconds-milliseconds)
- ADJ_FLD(SSSS.mSS) Sum of field time and following between time (seconds and milliseconds)
- EXIT “1” if last field in the form during a session; “0” otherwise

- METAFILE NAME Complete name and path for the *project.bmi* file (version of the instrument)
- METAFILE TIMESTAMP Metafile creation date and time
- WINUSERNAME Windows user name (*login* name on network)
- REMARK_CLK Number of times the interviewer opened the remark window
- REMARK_CHNG Number of times the interviewer entered or changed a remark
- ERROR Number of times there was a Blaise Action of Error Jump, Error Suppress or Error Escape.

Metafile name, *metafile timestamp*, and *winusername* appear only on summary lines (for example, FORM_TOTAL). Flags indicating whether a function was invoked at an item (“1”; “0” if not) follow these fields. Current functions are from Blaise Actions that are output to the .ADT or .ADK file. Future versions of the program will output additional flags, for both Blaise actions and keystrokes pressed (see final section).

Item, block, and form times. Item times in the report represent time spent at an item for a specific visit to the item. Block times are the sum of times for all visits to items within the blocks. There are three times reported for every field, block and form: the field time, the time between that field and the next field (these times may or may not match), and an adjusted time (the sum of the prior two times). Each time is reported first in an hours-minutes-seconds-milliseconds format (HH:MM:SS:mSS), and then again in a seconds-milliseconds format (SSSS.mSS). The two formats are provided to accommodate the import requirements of various data management and analysis software packages. Generally ISR uses the adjusted times in analysis of audit trail data.

Since an interview may be conducted across multiple Blaise sessions, *ADT_Report.exe* reports both FORM_TOTALs (individual accesses of a form) and ALL_FORM TOTALs (sum of all accesses). A form total represents total time spent in an interview in one Blaise session, and the all-form total represents total time spent in an interview across all sessions. Thus, the program will report two form times and one all-form time for an interview that was suspended and later completed in a second session. The all-form times are at the end of the file. Figure 6 provides a sample report that shows item times, as well as block and form times (*ADT_Report.txt*).

Interview length (form) report. The interview length is the sum of all entries to a form or case, across all visits to all items in the form. Figure 7 shows the interview lengths data file (*ADT_ReportIWLlength.txt*), which is generated during batch processing. The interview lengths are then transferred to a table in ISR’s SurveyTrak sample management system. The case ID is based on the Blaise primary key (sample ID for ISR studies).

Figure 6. Sample from Data File with Item, Block, and Form Times

```

100009 ! 00028! 01! 01! A_FEM.INTRO_A2! 00:00:00.880! 0.880!
00:00:00.050! 0.050! 00:00:00.930! 0.930! 0! ! ! ! 0! 0! 0
100009 ! 00029! 01! 01! A_FEM.GOSCHOL! 00:00:00.980! 0.980!
00:00:00.060! 0.060! 00:00:01.040! 1.040! 0! ! ! ! 0! 0! 0
{...}
100009 ! 99998! 01! 98! BLOCK_TOTAL:J_FEM.VERIFYBLOCK[4]!
00:00:01.590! 1.590! ! ! ! ! 0! ! ! ! ! ! 0
100009 ! 99999! 01! 99! FORM_TOTAL ! 01:28:26.910! 5306.910!
00:00:00.000! 0! 01:28:26.910! 5306.910! 0!
c:\b1proj\NSFG\female\work\female.bmi!
March 05, 2001 10:00:14 AM! 1009! ! !
100009 ! 99999! 99! 99! ALL_FORM_TOTAL! 01:28:26.910! 5306.910!
00:00:00.000! 0! 01:28:26.910! 5306.910! 0!
c:\b1proj\NSFG\female\work\female.bmi! March 05, 2001
10:00:14 AM! 1009! ! !

```

Case (sample) ID Block time
HH:MM:SS.MSS

Interview (form)
length
HH:MM:SS.MSS

Item time
SSSS:mSS

Figure 7. Interview Lengths Data Produced during Batch Processing of AuditTrails

```

PROJECT_ID, CASEID , IwLength(minutes)
TestProj, 100009, 65.1
TestProj, 100009, 48.1
TestProj, 100009, 76.6
TestProj, 100009, 77.3
TestProj, 100009, 108.0
TestProj, 100009, 109.0
TestProj, 100009, 67.8
TestProj, 100009, 95.8
TestProj, 100009, 68.7
TestProj, 100009, 52.0
TestProj, 100009, 71.3
TestProj, 100009, 57.2
TestProj, 100009, 71.4
TestProj, 100009, 74.8
TestProj, 100009, 79.1

```

1.5 Analysis

The current version of *ADT_Report.txt* provides the following information:

- Item and form times, and block times if requested
- Counts for each item of exits from the interview, remarks accessed, remarks entered or changed, and errors

In addition to automatically calculating and storing interview lengths, ISR uses audit trail files for the following purposes:

- Examining pretest form and item times, in order to determine whether survey instruments need to be shortened
- Identifying items at which at which interviewers overall may be experiencing a higher incidence of problems
- Identifying specific interviewers who may be experiencing an unusually high number of problems

The *ADT_Report.exe* data (exclamation point delimited) can be read into MS Access, SAS, SPSS, or any other data management or analysis software that can read ASCII delimited text. The following section provides examples from SAS analysis of the *ADT_Report.txt* file for .ADK files (times and keystrokes) from a large pretest for a study.

2.0 Findings

We analyzed data on audit trails (using the *KeystrokesISR.dll*) from a large pretest (619 interviews, 5,835 survey items, 30 interviewers). The pretest was for the National Survey of Family Growth (NSFG), Cycle 6, conducted during March through July 2001. There were two survey instruments, one for male respondents (n=310) and one for female respondents (n=309). Items with higher than average counts of item-level exits, errors (consistency checks), and average times were examined. We discuss each of these in turn, providing examples of potential problems in CAI instruments they can help identify.

2.1 High counts of exits from interview

The criteria for identifying pretest items with high interview exit rates were (1) that there were 200 or more visits and (2) that the exit rate for the single item (across all visits *to the single item* across all sessions in all interviews) was at least two standard deviations from the mean item-level exit rate *across all items*. Mean item-level exit rates were .05 (standard deviation = 1.8) for females and .07 (standard deviation = 2.2) for males. Table 1 lists the items that met both criteria⁶.

⁶ Less conservative selection criteria might have revealed additional problems. Any screen (other than the final screen) at which there is a relatively high number exits should be investigated for possible design or interviewer performance problems.

Table 1. Pretest Items with High Number of Exits from Interview

Item	Number of exits	Number of visits	Exit Rate (%)
Final screen in interview (female)	309	316	97.8
Final screen in interview (male)	310	318	97.5
Verify pregnancy outcome (female)	72	844	8.5
Introduction (section A female)	24	520	4.6
Introduction (section A male)	18	420	4.2

These items appear to represent three types of exits (1) normal exits at the end of the interview (male and female final screens); (2) getting into the first screen of the interview and then exiting without conducting the interview (introductions); and (3) abnormal termination of the interview at a verification item in the female instrument. There are reasonable potential explanations for the first two, the first benign, the second less so.

On the first, one might expect there to be a 100% correspondence between visits to the final screen in the interview, and exiting the interview. One explanation for less than 100% correspondence might be that interviewers in some cases backed up to review prior items before exiting. For example, the interviewer arrives at the final screen (“This concludes the interview. Thank you for your participation in this important study.”), backs up for some reason to a prior item, revisits the final screen and then exits the interview. The first visit would not be flagged for exit, but the second would. This happens in less than 3% of the visits, and would not be considered a serious problem with the final screen *per se*. Examination of keystrokes in audit trails for cases involved would contradict or confirm the hypothesis about backing up to prior items. If supported, it would be useful to know what items were the targets of the back up sequence. Planned enhancements to audit trail reports would allow us easily to obtain such information (see Section 3).

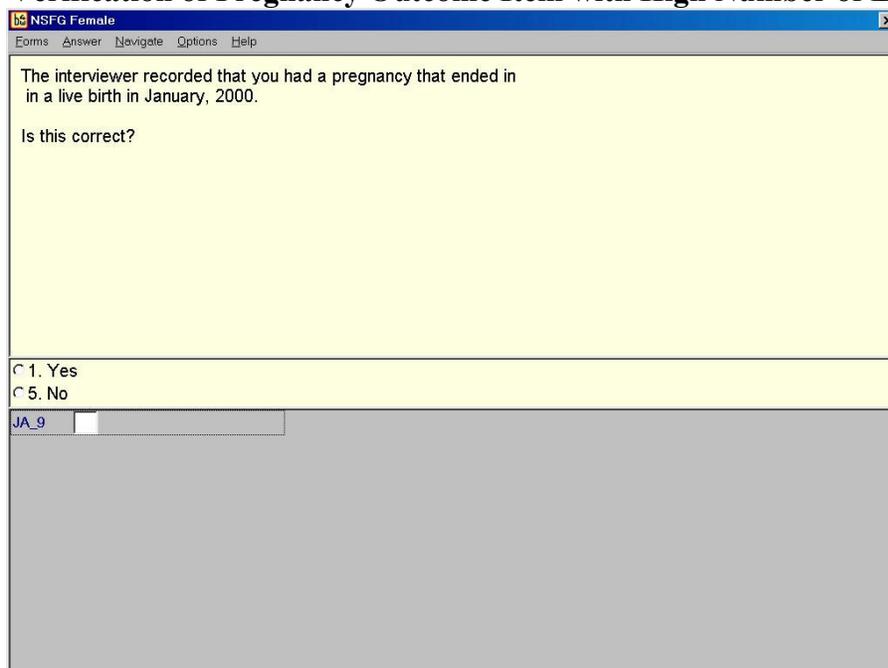
On the second, it appears that interviewers arrived at the first screen in the interview, and more than 4% of the time exited without continuing. This suggests that interviewers sometimes get into the interview without actually knowing that they are able to proceed with the interview. This may be a training problem, or an aspect of the logistics of the decision to continue and move from the sample management system to the survey instrument that has not been anticipated. Additional investigation is necessary in order to identify the reasons for this unusually high number of exits at the introductory screens. These are also items at which there are unusually high item times. This may suggest that interviewers that exit do not make the decision to exit immediately after entering the screen (see discussion about item times below).

We explored why there would be very high numbers of visits to these items (520 for females and 420 for males). Since interviewers always revisit the introduction screen on reentry to cases previously exited, we calculated the total number of exits across all items in all cases, and found it to be 842. This accounts for approximately 90% of the 940 exits, which suggests that interviewers occasionally move forward through the first few items of the section and then return to the introduction screen. These early items ask the respondent to verify or provide additional information about household members already listed in a separate screener instrument. Interviewers may be returning to the introduction screen in an attempt to return to the screener, which was prohibited in this study. Detailed examination of the audit trail files would determine

whether this is the explanation for the extra visits to the introduction screen. If this is the case, then the instrument designers could consider providing a mechanism for the interviewer to review a summary screen displaying the original roster, perhaps accessed via a function key.

The large number of exits at the verification item in the female instrument is more perplexing. These items occurred in a Blaise block used to verify the outcome of one or more pregnancies a respondent reported. The block occurred in a computer assisted self interview (CASI) section of the instrument that the respondent completed. The item (see Figure 8) included text fills, and by random assignment the respondent either read question text only or read question text while listening to text through earphones (Audio-CASI or A-CASI). Thus, for some respondents Blaise was also loading sound or .WAV files at this screen (a separate sound file for each segment of question text or text fill). Of the 72 verification screens from which interviews were abnormally terminated, 57% were text only and 43% were A-CASI.

Figure 8. Verification of Pregnancy Outcome Item with High Number of Exits



The screenshot shows a window titled "NSFG Female" with a menu bar containing "Forms", "Answer", "Navigate", "Options", and "Help". The main content area is yellow and contains the text: "The interviewer recorded that you had a pregnancy that ended in a live birth in January, 2000. Is this correct?". Below the text are two radio button options: "1. Yes" and "5. No". At the bottom left, there is a small input field labeled "JA_9" with a cursor.

Discussion with project staff revealed that respondents were required to type a “1” or “5” and press the [Enter] key at each pregnancy outcome verification item, and that they received reports that at least in some instances respondents experienced a delay after pressing [Enter]. Some respondents reacted by continuing to press the number and/or [Enter]s, and were eventually “thrown out of” the instrument. Thus, these 72 exits appear to represent abnormal terminations of the interview. Further investigation is necessary to determine whether these exits are related to the tailoring of text, the loading of multiple sound files, a software problem, how the verification block was programmed, or a combination of such factors related to a specific respondent. It is possible, through direct examination of the audit trails, to see exactly what keystrokes these respondents entered. This might assist in diagnosing the source of the problem at these screens.

In addition to providing information about survey items that may be problematic, audit trails can reveal information about interviewer behaviors. For example, we explored whether or not some interviewers were more likely than others to exit at the introduction item. Table 2 shows that only a few interviewers were responsible for the majority of exits at this item. Fourteen of the total of 30 interviewers exited at least once at the survey introduction screen. The percentage of the total exits each of the 14 interviewers was responsible for ranges from about 2.5% to 26%. Three interviewers alone (10% of the 30) were responsible for approximately 55% of the 42 exits.

Table 2. Individual Interviewer Percentages of Total Exits at Introductions

Interviewer Rank (high to low)	Number of cases	Number of exits	Interviewer exit rate (% of cases)	Percentage of total exits ^a	Cumulative Percentage
1	18	11	61.1	26.2	26.2
2	29	7	24.1	16.7	42.9
3	9	5	55.6	11.9	54.8
4	75	4	5.3	9.5	64.3
5	33	4	12.1	9.5	73.8
6	14	2	14.3	4.8	78.6
7	24	2	8.3	4.8	83.3
8	13	1	7.7	2.4	85.7
9	26	1	3.8	2.4	88.1
10	44	1	2.3	2.4	90.5
11	1	1	100.0	2.4	92.8
12	15	1	6.7	2.4	95.2
13	29	1	3.4	2.4	97.6
14	16	1	6.3	2.4	100.0
Total		42		100%	

^a Individual percentages do not sum to 100% due to rounding.

2.2 High counts of errors

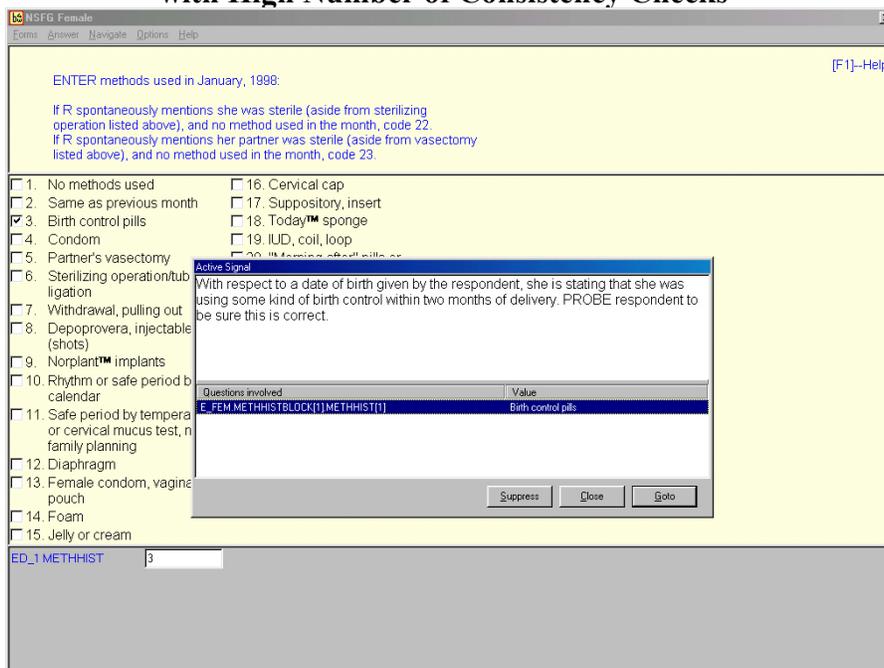
ADT_Report.exe provides an “error” flag for each item visited, which is “1” if any Blaise error action occurred (Error Jump, Error Suspend, Error Escape), which are the result of programmed Blaise consistency checks. There were 4,525 consistency checks across all items in all cases. One item alone, a block of items asking for birth control history over a 36-month period had 1,332 consistency checks over 10,723 visits (see Figure 9). That is, 12.4% of the time one of the blocked items was asked, interviewers were asked to check the consistency of the response against other items in the series and elsewhere in the interview. This represents approximately 29% of the 4,525 consistency checks across all items in all interviews.

A great deal of care went into the crafting of custom error messages and programming a large number of consistency checks for this section, in an attempt to improve the quality of data. On such items there seems to be a tradeoff, however, between higher data quality and increased interviewer and respondent burden. The project has decided to redesign this section, placing the

items in a calendar-like grid, in an effort to reduce both tedium and the number of consistency checks that occur, without reducing the number of programmed consistency checks.

Examination of mean case-level and interviewer error rates revealed that a few interviewers had much higher or lower mean counts of consistency checks per case. The mean count per case was approximately 8. One interviewer averaged 42 consistency checks per case over 5 cases, while another interviewer averaged zero consistency checks across eight interviews. Without further investigation it is difficult to provide an explanation for these outliers, which could be attributable to respondent characteristics, interviewer behavior, or both.

Figure 9. Birth Control Method History Item with High Number of Consistency Checks



2.3 High item-level times

Mean item time across all items was 12.1 seconds for female respondents, and 11.4 seconds for male respondents. However, using the same criteria as for exit rates (200 visits and two standard deviations above mean), we identified approximately 20 items with relatively high mean times, ranging from a high of 3 minutes 32 seconds to 38 seconds. Some of the items with high mean times parallel those found in the analysis of high exits: (1) the introduction screen (3 minutes 32 seconds mean time in the female instrument and 3 minutes in the male instrument); final screen (1 minute 30 seconds in female instrument and 1 minute 53 seconds in the male); and verify pregnancy outcome (approximately 1 minute in female).

Some interviewers were relatively higher or lower than the means on interview length. For example the mean female interview length was 2 hours and 13 minutes. The means for two interviewers (out of 30) were two and one standard deviation above the mean (4 hours and 52

minutes for 1 interview, and 3 hours 3 minutes across 3 interviews). The means for two other interviewers were at least one standard deviation below the mean (mean of 1 hour 33 minutes across 15 interviews, and 1 hour 24 minutes across 12 interviews, respectively). Further analysis could investigate the general level of correlation among certain behaviors (for example unusually high exits, item times, and consistency checks. Direct examination of audit trail files for outlier interviewers, cases, and items could help explain some of the high item times and identify questions to delete from the pretest instrument or questions to redesign in an effort to improve usability and interviewer efficiency. These findings also suggest that interview length may go down with experience on the survey and increases in number of interviews taken.

In summary, audit trails provide a wealth of automatically collected data that that can be used to identify potential problems at the question, interviewer, and case levels. With the added capability of capturing keystrokes as well as item times, Statistics Netherlands has now made it possible to learn a lot about some of the sources of problems, and the situations in which they occur (including item-level events and behaviors, and the sequences in which they occur). Improved reporting programs will make it possible to quickly report on these data and identify key problems in survey questions and interviewer performance.

3.0 Future steps in the development of audit trail analysis tools

ISR programmers have begun modifications to the basic *ADT_Report.exe* in order to report on additional functions or interviewer actions that occur at each item, and we are currently testing a beta version. In this version, function counts *at the item level* are:

- SUSP Number of times the interviewer suspended or exited the form normally
- ABSUSP Number of times the interviewer suspended or exited the form abnormally
- TOTSUSP Total number of times the interviewer suspended the form, either normally or abnormally

- C_X_SUSP Number of times the interviewer form suspended using the CTRL_X function
- REMCLK Number of times the interviewer opened the remark window (via menu or F2)
- F2_REMCLK Number of times the interviewer opened the remark window by pressing F2
- REMCHNG Number of times the interviewer entered or changed a remark
- QHELP Number of times the interviewer accessed question-level help (via menu and F1)
- F1_QHELP Number of times the interviewer accessed question-level help with F1
- BLAISEHELP Number of times the interviewer the interviewer accessed the Blaise Help function

- NEXTLANG Number of times the interviewer selected “next language” (via menu or F12)
- F12_NEXTLANG Number of times the interviewer selected “next language” via F12
- PREVLANG Number of times the interviewer selected “previous language”
- C_L_LANG Number of times language option box was opened at this field

These counts reflect recorded Blaise actions, pressed keystrokes, or a combination of both. Thus, they are most useful when using the *KeystrokesISR.DLL* to generate audit trails. The ability to distinguish between types of function access (e.g., menu versus function key) allows us to evaluate the effectiveness of training and interviewer performance in use of specific functions. For example we examined the frequency of help access for these pretest interviews and found that help was accessed very little across all items in all pretest interviews (item-level mean for females was .04 and for males .08). It would be useful to explore whether rates of help access were higher at the beginning of a study, whether some interviewers access help more than others,

or whether some items have much higher rates of help access. As with the analysis discussed in Section 2, examination of function use in this way may reveal problems or weaknesses and lead to improved design of screens, on-line help, training, and so on.

In the future we will add additional counts to the item-level data and new reports. One report we expect to create will be a “back up” report that will include items that were the target of a backup sequence (one or more keystrokes were pressed to return to a prior item), the source item (from which the backup sequence was initiated), the number of items backed up through, and the original and final data values for the target item. ISR was able to generate such reports for CASES and SurveyCraft, which were used to identify problematic survey questions (see, for example, Couper and Schlegel, 1998). Another report we hope to create will report on item-level error signals and checks, and items gone back to, with original and final answers at those items. A third report we have considered creating is a report on the actions and times associated with the use of multimedia files, such as playing and replaying video and audio files. Finally, we plan to report on item-level response latencies, and case- and interviewer- level response latency times, based on the difference between the times fields are entered and the time the first keystroke is pressed.

Current versions of the audit trail reporting program and DLLs have been placed in the International Blaise Users Group (IBUG) *E-Room*. The internet address is <http://eroom.isr.umich.edu/eroom/isrrooms/Blaise>, and the login and password are *Blaise User* and *bclub*, respectively (case sensitive). Access does not require that you have licensed *E-Room* software. As new versions become available, we will place them in the *E-Room*. Questions may be sent to sehansen@umich.edu or tmmarvin@isr.umich.edu.

4.0 References

- Bassili, John N., and Stacey B. Scott (1996). Response Latency as a Signal to Question Problems in Survey Research. *Public Opinion Quarterly* 60(3):390-399.
- Couper, Mick P., Sue Ellen Hansen, and Sally Sadosky (1997). Evaluating Interviewer Use of CAPI Technology. In Biemer, Paul, *et al.*, *Survey Measurement and Process Quality*. New York: John Wiley & Sons.
- Couper, Mick P., and Jay Schlegel (1998). Evaluating the NHIS CAPI Instrument Using Trace Files. *Proceedings of the Survey Research Methods Section*. Alexandria, VA: American Statistical Association.
- Marvin, Theresa, *et al.* (2000). Collecting Timing Data in Blaise. Report submitted to the U.S. Bureau of Labor Statistics (Task Order #22, Contract Number J-9-J-0025), June.

Session 9: Blaise CATI

- Integrating the Use of Data Centers into the NASS CAI Structure
Roger Schou, National Agricultural Statistics Service, USA
- Overview of Blaise at Iowa State University Statistical Laboratory
L. L. Anderson, J. M. Larson, D. G. Anderson, S. M. Nusser
Iowa State University, USA
- Scheduler Issues in the Transition to Blaise
Leonard Hart, Mathematica Policy Research, Inc., USA
- Considerations in Maintaining a Production Blaise CATI System
Kenneth P. Stulik, U. S. Bureau of the Census

Integrating the use of Data Centers into the NASS CAI Structure

Roger Schou, National Agricultural Statistics Service, USA

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 a Data Center in Cheyenne, Wyoming with 30 computers dedicated to CATI data collection. They also have an additional 22 computers that are used mainly for training, but 11 of them could be used for data collection, as there are 11 phones in the training room. NASS will open another official Data Center in Helena, Montana in early 2002 with 31 computers dedicated to CATI data collection, and 12 computers for training, which could also be used for data collection. Later in 2002, another Data Center will open in Louisville, Kentucky.

The Wyoming Data Center currently services four other states on a regular basis. The office in Montana services one state, performing Data Center calling activities utilizing their office computers.

In a typical state 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 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 majority of the Manipula setups used in the Data Center effort are generic from survey to survey. Therefore, coordination between Headquarters and the state offices is minimal, once they have received the applications. Most of the coordination is between each Data Center and their User States.

The basic functionality is relatively simple. The office with little or no CATI capacity (the User State) transfers cases selected for CATI to the Data Center. The Data Center collects the data for these cases and returns them to the User States. Staff in the User States then interactively edit the cases using Blaise. By returning the case to the User States for editing, we assure that all data in the state is edited in the same manner. A more detailed description of some of these processes will follow.

Each User State identifies the cases within their sample to be collected in CATI by the Data Center. Since the samples are usually available before the final version of the Blaise instrument, the ASCII files used to initialize the Blaise data set are zipped into a zip file. These ASCII files originating from the User States will only contain the cases to be collected by the Data Center. The zipped file is placed on the File Transfer Protocol (FTP) server and an e-mail message is sent to the Data Center's official mailbox. Each User State then initializes their entire sample into a Blaise data set. The cases corresponding to the ones sent to the Data Center are marked as non-CATI cases, by using a Manipula setup to compute a field in the management block. This field is then used to exclude these cases from the day batch.

Upon receiving the e-mail message from a User State, the Data Center moves the zipped file over the Wide Area Network (WAN). After unzipping the file, they concatenate the input files from the User State to the input files for their state. After they have concatenated each of the User State's files with their own, they initialize their Blaise data set, which will contain their own sample and all of the User States' CATI samples. Since all of the cases to be completed by the Data Center for a given survey are in one data set, managing the survey requires little additional effort on their part.

The zipping and unzipping processes and the e-mail message are all done within a Visual Basic menu script, so they are as easy as clicking a button. Currently, the only manual processes are the actual moving of the forms over the WAN, which is simply a "drag and drop" operation, and the concatenation of the User States' files with their own.

As the Data Center completes the User States' cases, they are sent back to the appropriate state on a daily basis. The Data Center clicks on the "Send CATI Completes Home" button on their CASIC interface, a Visual Basic interface. This process invokes a Manipula setup that creates a Blaise data set containing a copy of all completed cases that have not been previously sent. After it creates a copy of a case, it marks the process switch field in the management block in the Data Center's data to indicate that the case has been sent. The process continues by zipping the newly created data set and placing the zipped file on the FTP server. An e-mail message is then sent to the respective User States, notifying them that a new data set exists on the FTP server.

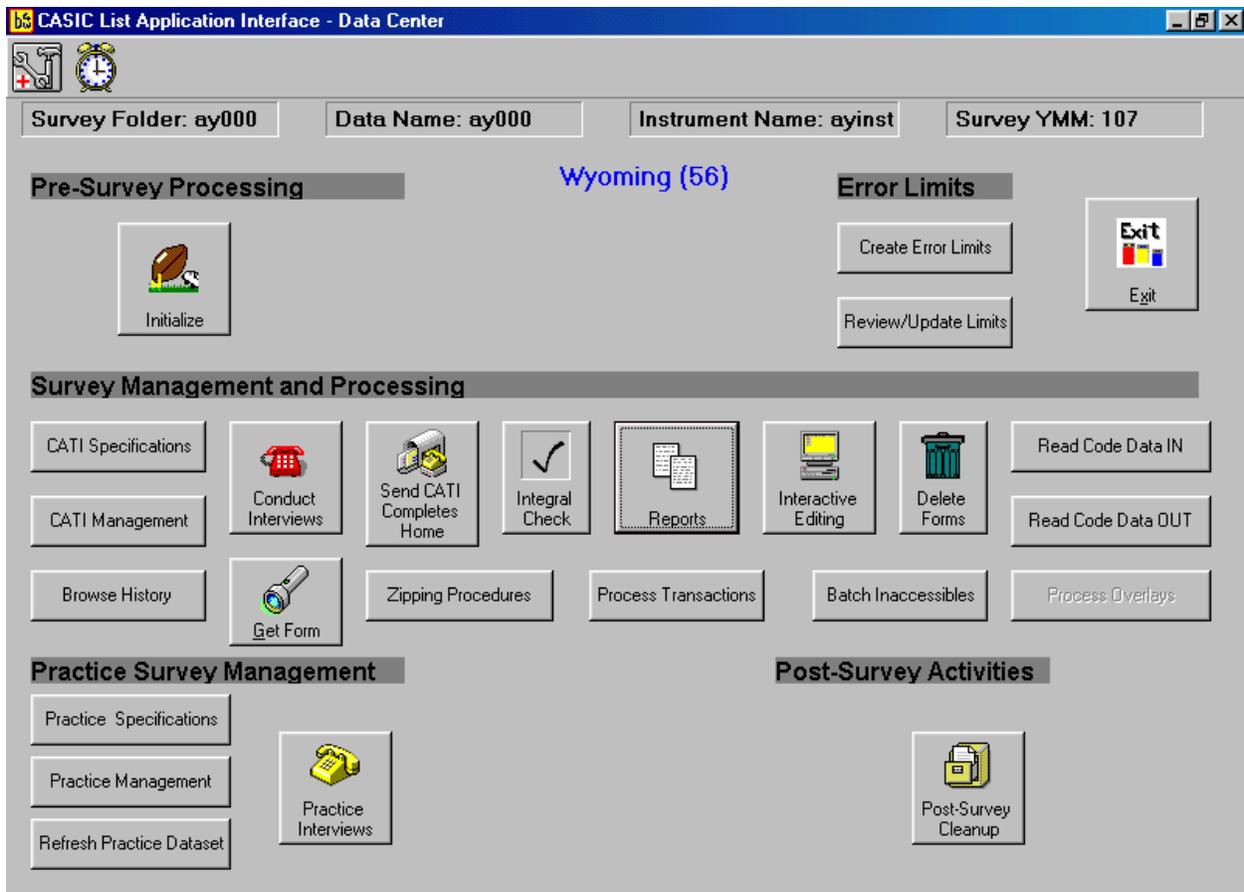


Figure 1 Data Center Interface (Visual Basic)

When a User State receives the message, they manually move the file over the WAN to a pre-determined folder. Then they click the "Merge Data from Data Center" button on their CASIC interface. This process unzips the data set into a temporary folder. Next a Manipula setup runs that locates the case in their data set and checks to see if it still has no data. If it has no data, the case is deleted from their data set, and then copied from the data set received from the Data Center. If for some reason, the User State had entered data for a case that was coming in from the Data Center, the Data Center's case is written to an overlays data set.

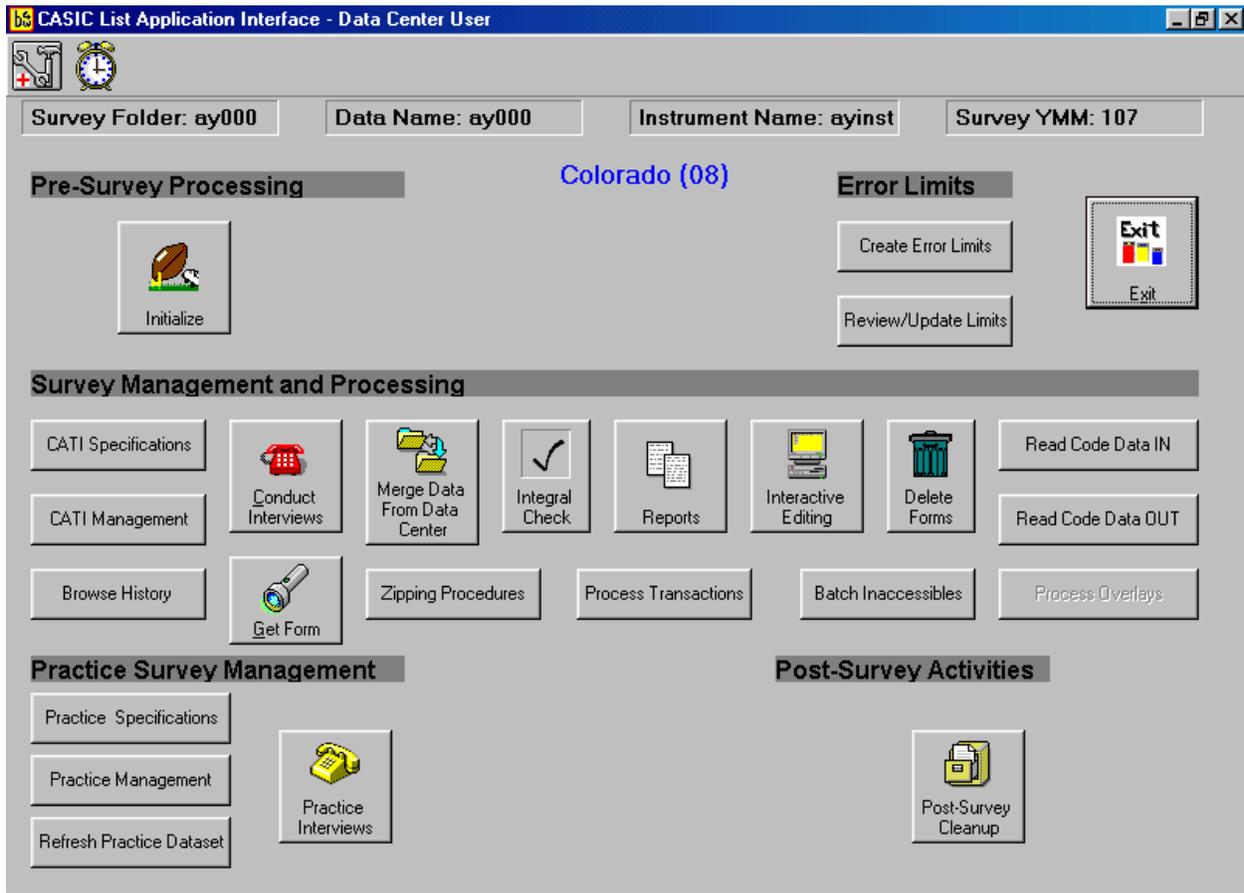


Figure 2 User State Interface (Visual Basic)

If an overlays data set exists, a ManiPlus setup can be invoked from the "Process Overlays" button on the CASIC interface. This setup will display a table of key fields for each case in the overlays data set. Upon selecting a case, a pre-determined set of fields from will be displays showing data from the case in the overlays data set and the survey data set. There are also buttons that will allow a user to start the data entry program for either case. Once a decision has been made as to which case to keep, the appropriate case is written to the survey data set, and the other one is deleted. Once a case has been resolved, it is removed from the table listing the key fields. This process can be repeated until the table is empty.

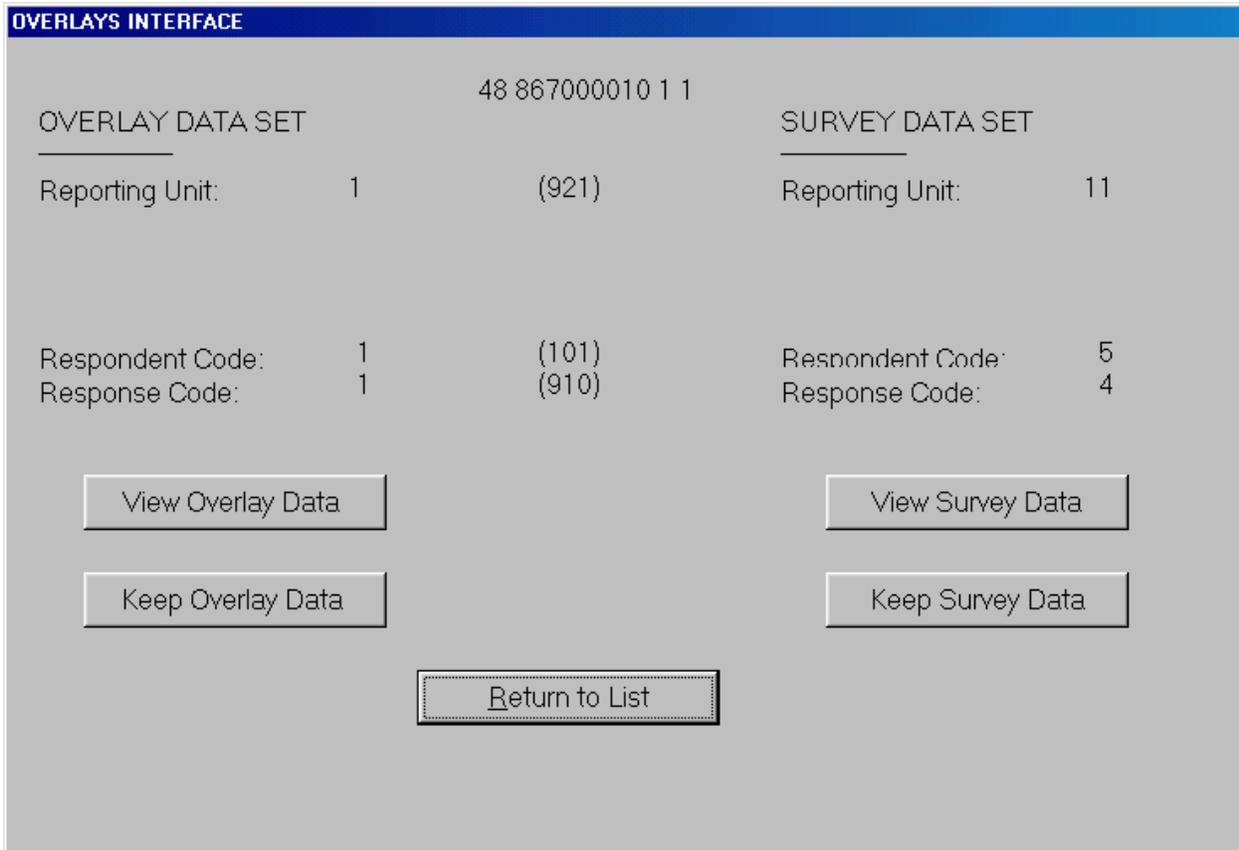


Figure 3 Overlays Interface (ManiPlus)

Except for the addition of the merge and zipping buttons, the User States' interface looks the same as it would for any other survey. All of the other processes run as they normally do in the User States, sharing the same set of Manipula setups. The interfaces for a typical survey, a User State, and a Data Center are very static, as seen in figures one and two.

The Data Center's interface also has just a few extra buttons (send completes, zipping, and a special report), however, many of the Manipula and ManiPlus setups running the other survey processes have additional code in them. This is due to the different handling requirements between a case that "belongs" to the Data Center's state, and a case the "belongs" to a User State. For example, the Data Center does not interactively edit a case that "belongs" to a User State. However, the data collection and interactive editing processes are all conducted in the same data set. The same data set that contains the Data Center's cases and the User States' cases. Thus, the ManiPlus setup, that runs the interactive editing process in the Data Center, filters out all cases except the ones that "belong" to the Data Center state. The same is true for the integral check and read out processes.

Many of the Manipula and ManiPlus reports the run in the Data Center were altered to either ignore all states but their own, or include extra reports for the User States' cases. For example, the status report, a Manipula setup that gives a report of all cases in the data set, normally provides distribution counts of cases to be completed and statuses of completed cases. If the survey being conducted has been designated as a Data Center activity, the status report gives the same information, but it is by state. It also has a breakdown of the data set as a whole for all states.

Another example is the disconnected phone report. The Manipula and ManiPlus setups that run to complete this process create a listing of cases where the phone number of the respondent has been disconnected. This listing only contains cases from the Data Center's state. Other files are created, one for each User State that has disconnected phone numbers in the data set, as well as a file of parameters for the e-mail package used by NASS. The Visual Basic menu script then sends an e-mail message with the attached file of that particular state's disconnected phones. The current NASS thinking is that if a Data Center encounters a disconnected phone number, the case becomes the responsibility of the User State to which it belongs.

NASS has been using this Data Center / User State relationship since September 1999 with much success. Some tweaking of the system was done during the early stages of implementation, as expected. However, the system matured rather quickly, and has been quite successful.

Overview of Blaise at Iowa State University Statistical Laboratory

L. L. Anderson, Iowa State University Statistical Laboratory

J. M. Larson, Iowa State University Statistical Laboratory

D. G. Anderson, Iowa State University Statistical Laboratory

S. M. Nusser, Iowa State University Statistical Laboratory

Introduction

In 1998, Iowa State University Statistical Laboratory became one of the first university users of Blaise in North America. The Statistical Laboratory (Stat Lab) began collecting data using Computer Assisted Telephone Interviewing (CATI) in 1995, but was interested in finding a different product, one that could meet a small shop's need for simplicity while providing the ability to conduct large and complex studies.

After previewing a number of software options, the Stat Lab selected Blaise for several reasons: its ability to handle large complex data sets; its ease of use for both interviewers and supervisors; its strong data editing and case management capabilities; it would soon be in a Windows version; and the prospect of strong user support from Westat.

In this paper, we outline our initial experiences with Blaise and identify features that we hope to explore in the future.

Description of Statistical Laboratory

The Stat Lab's survey research data collection unit is small (20 CATI workstations and five professional staff, including one programmer). CATI projects typically require shifts of one to eight interviewers. Interviews are conducted primarily during evening hours, with a lesser amount of day and weekend contacts made as required by the project. Smaller projects can include some day and weekend shifts that are not staffed at all, while larger international projects might require virtually round-the-clock interviewing.

The nature and scope of projects varies tremendously. The Stat Lab has conducted numerous survey research projects with complex respondent selection criteria, extensive rostering, and intricate questionnaire routing with sub-interviews. Clients include university administration, faculty with research funded by external organizations, other universities, government agencies, and other non-profit organizations. Research topics cover issues such as welfare reform, business development, agriculture, economics, education, and health. Mail and web surveys, personal interviewing, and focus groups are all conducted by the Stat Lab.

Case management in a small shop

We first used Blaise on relatively simple projects, a marketing study for the university student newspaper and an administrative study of university retirees. The questionnaires were straightforward and the target populations (students and retirees) were easily contacted list samples. Although the questionnaire itself performed well, our staff received the first indications that adaptations were needed for efficient case management. Specifically, we needed more extensive contact and background information available to

interviewers, and we needed to readjust the parameters in the CATI Specification Program since various cases sometimes appeared to interviewers at inappropriate times.

Our highest priority subsequently was to make a complete call history available to interviewers. Each dial result is now written to the database. These results, along with appointment and other information, are displayed on an infopane before the start of the interview (Figure 1). The call history screen is adapted for individual project needs. It provides interviewers with sufficient information about the case to increase their comfort level and to facilitate wise decisions.

To ensure that interviewers view the history screen before dialing, only the Questionnaire option is available on the Make Dial screen. All dial results are available within the questionnaire as parallel blocks in the tab format. After the introduction screen is passed, the dial results are no longer available.

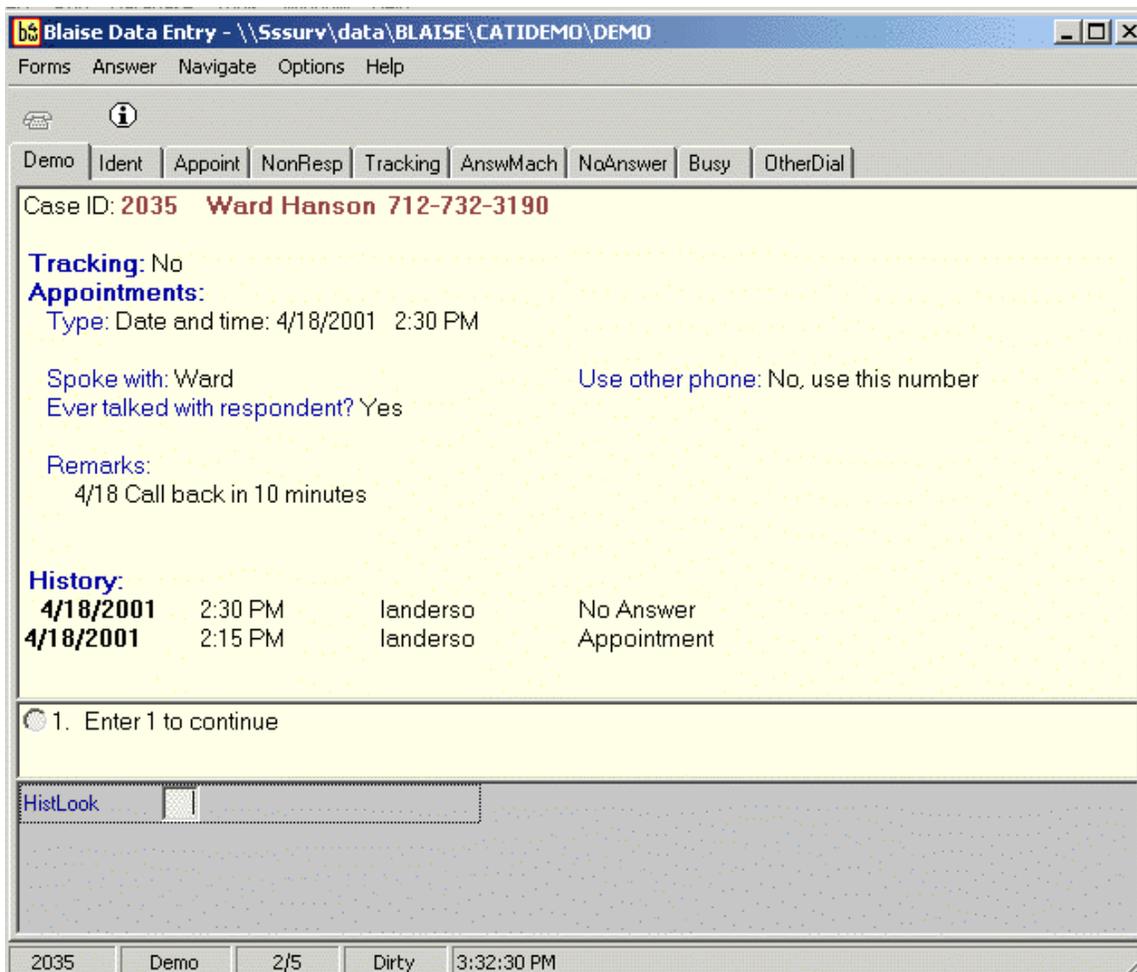


Figure 1. Call history screen.

Our difficulties in controlling the autoscheduler increased with a business development study, which indicated again that we needed to learn more about how to set parameters, how to handle various types of callbacks, and how to add another level of control when we have prior knowledge of when to call. For

example, most types of businesses should be called during daytime business hours, but a restaurant should not be called near lunchtime.

Callbacks on this study became a problem when we did not talk to the respondent (the manager of the business), but to an employee gatekeeper who would inform us that “He’s out of the office this week”, or “You can probably get her late in the afternoon”. We ended up with many soft priority cases that repeatedly came up before untried cases. Also, if the soft priority cases did not get a response within the period set, they would not come up again for a long time. We wanted a more even distribution between these cases, for which we had no real appointment information, and cases that had not been tried at all. Additionally, hard appointments that were not met sometimes surfaced inappropriately the following morning.

Increasing the number of days between no answer calls and reducing the maximum number of dials helped bring up more untried cases. We used interviewer groups to assign one interviewer only fresh numbers. We also trained the interviewers to make appointments more specific or more general, as appropriate. We need to look into methods to specify times to call for specific forms, before calling. Assigning different time slice sets to different types of businesses might help us call at appropriate times, but we need to make sure we always have someone working during all time slices. Our initial attempt to use time slices was unsuccessful because we did not always have someone scheduled to work during each time slice.

Our first welfare reform study used a list sample of welfare recipients. Because it was expected that a great deal of the contact information from the frame would be incomplete or incorrect and extensive tracking would be required, the autoscheduler was not used. All locating was done by telephone, using references such as National Change of Address (NCOA) and cross directories. Because supervisors did most of the tracking, and because most of the information they received from calling was vague and not easily classified, it was easier to have the information for each case on paper record-of-calls (ROCs).

In a later welfare reform study, we used the autoscheduler rather than the paper ROCs, making use of a five-row tracking table to store information from five tracking attempts (Figure 2). This information includes old telephone numbers, new telephone numbers, the source of the new telephone number, and an open field for any information available. If a new number is not known, any information that the interviewer has may be entered, and that information will be available in reports. If a new number is known, the number can be tried immediately. The new number replaces the old telephone number in the original telephone number field used in the Make Dial screen.

Blaise Data Entry - \\Sssurv\data\BLAISE\CATIDEMO\DEMO

Forms Answer Navigate Options Help

Demo Ident Appoint NonResp Tracking AnswMach NoAnswer Busy OtherDial

What is the source of the new telephone number?

1. Person at previous number
 2. DHS
 3. Internet
 4. Directory assistance
 5. Respondent called in
 6. Respondent replied by postcard
 7. Other

	TrackQ	OldPhone	NewPhone	Source	MoreInfo	CallNewPhone
Track[1]	2	319-386-2480	319-385-2211	1		
Track[2]						
Track[3]						
Track[4]						
Track[5]						

2145 Tracking 1/1 Dirty 2:27:08 PM

Figure 2. Tracking block

All cases in need of tracking were written to html reports, with each case linked to a page giving all information collected in the tracking table. These pages were printed and given to the two interviewers we had in the field to locate respondents. The field interviewers conducted the interview with pencil-and-paper if they were able to speak to the respondents, or they provided the office with a new telephone number when they were unable to conduct the interview in person.

In this study we used a dual frame sample, with a list of welfare recipients to target the welfare population and an RDD sample to target both the general population and low income respondents who were not on welfare. A different set of dispositions was required for the RDD and list portions of the study. We also made use of an external file to code car make and model in a roster of household vehicles.

Finally, our most complex Blaise study to date has been the Family Business Research Project. Interviews were conducted first in 1997. All 1997 respondents were contacted again in 2000, this time using Blaise. The study was conducted for a consortium of researchers from 16 universities in the U.S. and Canada. Data were collected both about the business and how the business affected the family. Accordingly, both the business owner/manager and the individual responsible for managing the household were interviewed. In some cases, this was the same person. In 2000, we were thus attempting to contact either one or two people.

Routing for the questionnaire depended on whether the business was still owned or operated by the business manager respondent, and whether the business manager and household manager were still in the same household. If the business was no longer in operation or managed by our respondent, a short series of questions was asked. If the business manager and household manager were no longer in the same household, the business manager would be asked all business and household questions (a combination questionnaire), and the household manager would have a short series of questions. If the household manager was now the business manager, the household manager would answer the combination questionnaire.

Because of the difficulty of making appointments for two respondents who may no longer live in the same household, we used paper ROCs rather than the autoscheduler. We needed to have one form for both respondents because the preliminary part of the questionnaire determined the routing for both respondents, and the next part of the questionnaire (the household roster) was answered by the first respondent interviewed.

Data Management

The Blaise CATI Management reports and the edit mode are important to our case management. Because we may not have a crew working during the daytime on a small project, it is necessary for supervisors to be aware of upcoming appointments. The appointments graph and other reports in CATI Management make case management efficient by making it easy to keep track of appointments and the status of all cases.

The edit mode makes it possible to check the quality of the data and make changes immediately after an interview. Supervisors can access the production data model in edit mode with CATI disabled, using a shortcut icon.

For editing and coding, we use a new data model. This new data model includes additional codes for enumerated fields which have a value of “other” and are followed by an open field to specify the “other” answer. It also includes additional enumerated fields to code other open fields. The supervisors use a Manipula setup to read data into the edit data model. They can edit and code data while data collection is still in progress, thus shortening the data delivery time.

Use of Reports

The reports in Cati Manager are supplemented with reports run out by a Manipula setup and processed by SAS in a batch file at night. Summary reports are written to a text file and printed out; reports dealing with individual cases with appointments (Figure 3), cases with dial results such as refusals, nonresponse, other dial, tracking, and needs reset, and cases which have been called at least eight times are written to html files which are viewed in a browser and printed as necessary. Information in the reports includes case ID, interviewer, date of call, comments, and other appropriate information.



Figure 3. HTML Current Appointments report

Finally, the output data set is processed to fill in blank fields in the data and conform to the standard Stat Lab format. The Cameleon SAS script has been modified to write SAS statements to fill in empty fields and change the coding for refusals. All “not applicable” results are filled in with 8s, “don’t know” with 9s, and “refuses” with 7s.

Summary

In the future, we hope to refine our use of the case management features already available to us, such as appointment parameters; to explore options such as using select and sort fields and time slices; and to learn how to use Manipula to run updates in management fields.

There are some refinements we would like to see in the CATI Specification parameters, including the ability to set a different maximum number of dials for cases with default priority and for cases with appointments. Similarly, we would like to set a different number of minutes between no answer for cases with default priority and for cases with appointments. We would also like to set a different route back option for appointments and for other cases, such as refusal conversions.

Blaise has been very well received at the Stat Lab. Interviewers, some with minimal computer skills, find it very easy to work with. Supervisors are very satisfied with the editing and coding abilities and the possibility of conducting complex projects in Blaise. Data are processed quickly, allowing projects to be

finished quickly. It has been critical to have Blaise's efficient routing tools, which enabled us to create questionnaires with extremely complex skip patterns.

Overall, we have found Blaise to fulfill our requirements of simplicity while having the capability to do complex work.

Scheduler Issues in The Transition to Blaise

Leonard Hart, Mathematica Policy Research, Inc., US

Abstract

Mathematica Policy Research (MPR), a UNIX CASES user, has added Blaise 4 Windows as a CATI platform. One of the major issues in this transition is the standard Blaise Scheduler. In this paper I will describe MPR's scheduler needs, where the Blaise scheduler meets those needs, and where work-arounds need to be developed. The paper will more broadly touch on scheduling issues in a multi-platform environment, and the Blaise Inherit CATI mode in general.

Introduction

The Blaise call scheduler or, as referred to in the Blaise Developer's guide, the "CATI call management system," is a powerful tool in controlling the delivery of cases to interviewers. One simple line of code, "Inherit CATI", included in your program, takes this highly complex procedure and converts it into a easy to use parameter driven system. Ease of use comes with a cost however; it limits you to a given format for CATI operations. As we all know, no two surveys are exactly alike and each has its own special needs. This problem is compound by diverse procedures among survey organization. In this paper are a few brief examples on how we at Mathematica Policy Research (MPR) looked outside the box to use the call scheduler to meet our needs. Our goal was not to recreate the Blaise CATI management process but to enhance it.

Background

As Mathematica Policy Research looked into converting to Blaise, one of the deciding factors was the CATI Call scheduler process built into the system. As we started to gain experience with the Blaise family of applications, we started to note the limitations of the CATI call management system. At this point we sat down as a group at MPR and tried to decide how can we overcome these obstacles. Basically we had two choices; write our own CATI call management system or try to build on the Blaise CATI call management system.

Building our own system would yield a CATI management process that could allow us to customize the CATI management system as needed to meet specific survey requirements. The second choice was to use the Blaise CATI call management system and see if we could enhance it to meet each survey's requirements. We opted to go with the second choice once we started to look at the complexity of trying to develop and maintain our own CATI call management system. One of the key factors in our decision was based on our past experience with our other CAI system. We also felt that the introduction of the Blaise Component Pack, along with Manipula, Maniplus and Delphi, could help us meet our goal.

Call History

One of the first major challenges we encountered using the call scheduler was the need to maintain a complete call history of each case, along with other key information that we felt necessary. The current Blaise CATI call management system only keeps a call record of five attempts; the very first contact and the last four attempt. The call history between these attempts is lost. We needed the capability to keep all

attempts. This information is used in creating productivity reports, management of the survey and on help in addressing problems as the survey progresses. Figure 1 depicts the scheme currently kept by the call scheduler.

Figure 1.

Block TCall

WhoMade "who made last dial in call ": STRING[10]

DayNumber "daynumber relative to FirstDay ": 1..999

DialTime "time of last dial in call ": TimeType

NrOfDials "number of dials in call ": 0..9

DialResult "result of last dial in call ": TcallResult

Endblock

An additional problem with the current Blaise CATI call management history concerns the type of data being kept. We wanted to keep additional information such as end time of the call, current case status and interviewer notes to help us manage the survey process and to track interviewer performance. Our first thought was to modify the CATI.INC file by increasing the size of the array for Regscalls (see figure 2) so that it would keep track of all calls. This is the block of code the Blaise CATI management system uses to store information. Although increasing the array size for Regscalls worked, we had some concerns. First, what effect would modifying CATI.INC have on the Blaise system? The Blaise development team felt that this would not create a problem with any of the other Blaise applications. Second, as we upgrade to a newer version of the Blaise system it would require us to make changes to CATI.INC again. It would then be necessary to verify that these changes would not affect the other Blaise applications.

Figure 2.

FIELDS

NrOfCall "number of calls" : 1..99

FirstDay "date first call" : DATETYPE

Regscalls "registered calls" : **ARRAY[1..5] OF TCALL**

The next question was how to get the additional items of information we want to retain included in the CATI.INC file. This was a harder task. Our first attempts to save this additional information to the modified CATI.INC resulted in failure to write the information to this block at the proper time. We also needed to ensure that we didn't re-update our additional call information as an interview progressed. And once again, going this route meant that more changes would have to be made with each new version of Blaise. The more we worked on this solution, the less satisfactory it seemed to our problem. After some research we came across a prepared for presentation but not presented at the May, 2000 Blaise user conference in Ireland by Linda L. Anderson, Statistical Laboratory at Iowa State University.

Anderson's paper, "Displaying a Complete Call History to Interviewers", addressed the same problems we were facing; how to keep a complete call history and displaying the information for the interviewers to use. In her paper she looked at two different approaches to accomplishing these goals. The first approach was adding the key information to the history file. The disadvantage of this according to Anderson, is that the necessary information would not be available in a timely fashion. A process would have to be run against the history file to populate the information into the external Blaise dataset. For this process to work efficiently and avoid corruption of the external data file, all users and applications accessing the dataset would have to be suspended for the duration of the effort. Anderson's second approach involves the construction of a block of code to be included within the Blaise dataset to maintain the additional information. We choose this basic approach and expanded upon it.

In Anderson's approach, information is displayed once the case is loaded, after the dial screen. We wanted to display this information on the dial screen before the call was made. This format would allow interviewers a chance to review the complete call history with any interviewer notes before initiating the call directly on the dial screen (see appendix A for example of the screen). We achieved this by condensing and reformatting our own history block (BCall) to a field called CallHistNotes (see appendix B for example of the Blaise code). In this field we include who made the call, date of the call, the dial time, current status and the first 50 characters of the note field. We condense the note field to 50 characters to make the screen more user-friendly, since our note fields are defined as a STRING[32000]. On the dial screen after the CallHistNotes field, we also include the unedited note fields, allowing the interviewer to read the full notes if they like. The interviewer would normally have to scroll down to see this information.

By creating our own CATI call history block, we successfully overcame our problems with the default Blaise CATI call history block; however, we were getting multiple entries for each session in the start. We wanted to move the call history information down one level in the array after leaving the dial screen so the latest information is always the first position of the array. We accomplished this by using the insert(1) to our array, a solution that was initially problematic. Every time information changed in the CallHist block everything was moved down one level, resulting in multiple increases in our CallHist block. We also discovered that merely loading the case to the dial screen also caused the information to move down one level. Further more, when the case was brought up in conduct interview mode, the information in the array was shifted down once again. We overcame both of these problems by setting an AUXFIELD TempID to 1 (see figure 3) after the first Insert(1) was done. We used the field TempID as an indicator to tell us if the insert command had been incremented. Since the TempID field is not on the route, we had to use the KEEP method to maintain its value.

Figure 3.

```
TempID.KEEP
IF TempID = EMPTY "" THEN
  CaseHist.INSERT(1)
  TempID := 1
ENDIF
```

These modifications to our own call history block have basically met our basic needs although some refinements are needed to the process. The interviewers would like to see more information on the dial screen, but the current layout of the dial screen limits the amount of information we can display. Therefore, our next goal is to write a popup screen that would allow for us to display more information on

the screen in a user-friendly layout before the interviewer makes the call. With the introduction of the Blaise Component Pack, we feel we can write a Visual Basic program to clearly display this information.

Complex includes/excludes in daybatch

One of the options when creating a daybatch is to include or exclude cases based on a field in the Blaise dataset using the CATI Specification program. This works well and would be easy to implement if inclusion and exclusion needs were uniform across instances. Unfortunately, in some cases, complex exclusion or inclusion rules are necessary. For example, cases may be excluded based on a formula. The current CATI specification program does not allow such calculations.

We overcame this problem by creating a field in the Blaise dataset called CaseOnOff with a value of zero or one. Before we run the CATI Management program to create the daybatch, we first run a Manipula program to set the value of CaseOnOff to a zero or one, based on multiple criteria. The Manipula code listed in appendix C is an example of how we reactivate a case after a given number of days based on several fields. In this example we first look at two different fields to see if they meet a certain criteria; if so we then calculate how long this case has been on held status. If it has been last then fourteen days we turn the case off by setting the CaseOnOff field. Then in the CATI specification program we can include or exclude cases based on the CaseOnOff field.

Auto Dialer

When our interviewers make a call, a billing code must be entered before they get a dial tone. Each survey has its own billing code. Since our interviewers use a modem to make the call, we needed a way to pass along the billing information through the modem to the PBX system. Several options were available to us. One such option was to attach the billing code to a phone number prior to populating the sample information into the system. Another recourse involved direct modification of the dialer options.

Attaching the billing code to the phone number raised several problems. For example, if the interviewer needs to change the phone number, it would risk a corresponding change in the billing code when making the phone number changes. What if the interviewer was entering a brand new phone number would they remember to include the billing code; if so, would it be the right one. The second problem was if you had multiple phone numbers you would need to change the billing code for each of them. Our conclusion by adding the billing code to the phone number would increase the chance for errors.

The second option of setting the dialer options imposed its own problems. Here the problem is each machine might be used for different surveys, each having a different billing code. The question became how to set the billing code for each survey. This process was further complicated by our setup of Windows 2000, which limits users capabilities to implement modifications, thus making it more difficult for programmers to accomplish changes behind the scenes.

Our first approach to tackling this problem was including code in our Visual Basic menu system that would pass billing information for the dialer to the registry. As mentioned previously, the interviewer rights are limited on the machine and one of these limitations is the ability to update the registry.

The next approach involved the use of a macro scripting language called Winbatch. Winbatch allowed us the ability to script the needed changes to the dialer options. Our WinDial (see appendix D) script is generic enough to allow us to pass parameters through to it by our Visual Basic interface, thereby allowing each survey to set the proper PBX codes for the dialer. This process is done by executing WinDail twice through our Visual Basic menu system. During the first execution of WinDial the Visual Basic menu calls WinDail and sets the proper dialer options for a given survey. The second time, it hides

the dialer from the taskbar. Under normal Blaise calling circumstances, the dialer is minimized to the taskbar, but we found that this was confusing to our interviewers. We could have hidden the taskbar but that would have led to other problems. Hiding the dialer from the taskbar eliminated this problem.

Conclusion

As the Blaise community grows, more demands will be placed on the Blaise development team to expand the capabilities of the Blaise CATI call management system. Like any development team they have a limited amount of resources to apply to this area. I urge us as a community to work together to compile a list of needs that we feel that should be included in the scheduler and then present these items to the Blaise development team. Past experience with the Blaise development team has been that if modifications are desired by the community, most likely they can be built into the system. If not a reason will be given for its lack of feasibility.

In the meantime, I urge you to think outside the box to make the CATI management system work for you. You will be surprised what you can accomplish with the Blaise family of applications. In addition, with the recent release of the Blaise Component Pack and other programming and scripting languages, you should be able to do about anything you want to enhance your CATI management system without rewriting the Blaise CATI call management system. We all like turnkey applications, but as mentioned previously no two surveys are alike. Hopefully, through these examples and by working together as a community, we can make the Blaise CATI call management system fit anyone's needs.

References

Anderson, L (2000). Display a Complete Call History to Interviewers. Proceedings of the Sixth International Blaise users conference, Kinsale, Ireland, CSO Ireland.

Appendix A

Make Dial
✕

Dial menu

<input checked="" type="radio"/> Start Interview <input type="radio"/> No answer <input type="radio"/> Busy <input type="radio"/> Answering machine or service	<input type="radio"/> Non-working/Cell/Wrong number <input type="radio"/> Appointment <input type="radio"/> Non response
---	--

Questionnaire data:

SchoolPhone	(123) 456-7890
SpecialPhone	*
HomePhone	*
CaseID	110013
Respondent	Blaise Pascal
SchlName	USA Middle School
CDSP	
TCDSP	83
NumStudLd	4
BestTime	
CallHistNotes	MBrickel 6-22-2001 10:20 AM 83 gk said msge was picked up MBrickel 6-20-2001 11:23 AM 83 left msge w gk school closed DAbdelhamid 6-13-2001 1:57 PM 80 cb set if not rec'd GBuckley 6-11-2001 1:04 PM 80 spoke with Miss Pascal she said she mailed in in GBuckley 6-11-2001 9:27 AM 80 left toll free # and teacher ID at front desk ASherring 6-8-2001 4:43 PM 80 R says the Q's are ready to mailSome teachers hav ASherring 6-8-2001 3:56 PM 0 ASherring 6-8-2001 3:48 PM 0 CShering 5-29-2001 3:02 PM 80 R says the Q's are ready to mail SMale 5-24-2001 10:50 AM 80 best time to call is either 9-10am or 2-3 pm

Appendix B

BLOCK BCallHist

BLOCK BCall

FIELDS

WhoMade : STRING[20]

DailTime : TIMETYPE

EndTime : TIMETYPE

DailDate : DATETYPE

IntNotes : STRING

TCDSP : 1..999

RULES

WhoMade.KEEP

DailTime.KEEP

DailDate.KEEP

EndTime.KEEP

IntNotes.KEEP

TCDSP.KEEP

ENDBLOCK

FIELDS

CaseHist : ARRAY[1..30] of BCall

CallHistNotes : STRING[32760]

AUXFIELDS

TempID, I, J : INTEGER

LOCALS

K : INTEGER

RULES

I := 1

j.KEEP

j := 1

FOR K := 1 to 30 DO

CaseHist[K].KEEP

IF CaseHist[k].WhoMade = RESPONSE "" THEN

CallHistNotes := CallHistNotes + CaseHist[k].WhoMade + '' +

DatetoStr(CaseHist[k].DailDate) + '' + TimetoStr(CaseHist[k].DailTime) + '' +

STR(CaseHist[k].TCDSP) + '' + SubString(CaseHist[k].IntNotes, 1, 50) + '@/'

ENDIF

ENDDO

If I = J "" THEN

TempID.KEEP

IF TempID = EMPTY "" THEN

CaseHist.INSERT(1)

TempID := 1

ENDIF

CaseHist[1].WhoMade := StafInfo.User

CaseHist[1].DailTime := StrtTime

```
CaseHist[1].DailDate := sysdate
J := J + 1
ENDIF
ENDBLOCK

FIELDS
CallHist : bCallHist
```

Appendix C

AUXFIELDS (GLOBAL)

CurrentJulian : INTEGER

PROLOGUE

CurrentJulian := Julian(SYSDATE)

MANIPULATE

IF (Mangmnt.CDSP = EMPTY) THEN

IF (CallHist.CaseHist[1].TCDSP = 21) THEN

IF Julian(CallHist.CaseHist[1].DailDate) + 13 > CurrentJulian THEN

Mangmnt.CaseOnOff := 1

ELSE

Mangmnt.CaseOnOff := 0

ENDIF

ENDIF

ENDIF

Appendix D

```
; A Windows dialer, based on HelpDial.wbt
IntControl(50,0,0,0,0) ; Turn off Web Page Support
IF Param0 != 1 && Param0 != 3
    CR = StrCat(num2char(13),num2char(10))
    msg = WinExeName("")
    msg = StrCat(msg, " - Version: ",FileTimeGet(msg))
    txt = "Called with 1 parameter (/I or a phone #), this works as the
        Watcher/Dialer.%CR%With 3 parameters, it acts as SetDial%CR%%CR%SetDial Options:   LocalAreaCode
        Message(msg,StrCat(txt,"Note that these options are all upper case and are mutually
        exclusive%CR%%CR%If the first arg is a phone # and contains spaces, be sure to quote it!"))
    Exit
ENDIF
IF WinExist("Phone Dialer") == @FALSE
IF WinState("") == @HIDDEN THEN
    RunHide("Dialer", "")
ELSE Run("Dialer", "")
    WinWaitExist("Phone Dialer",-1)
ENDIF
IF Param0 == 3
    SendKeysTo("Phone Dialer","!td!e!c%Param1%{TAB}%Param2%{TAB}%Param3%~")
    WinClose("Phone Dialer")
    msg = "Dialer Parameters have been set..."
    goto theEnd
ENDIF
IF "%Param1%" != "/I"
    ClipPut(Param1)
    SendKeysTo("Phone Dialer","!n{Del}+{Ins}~")
ENDIF
IF WinExist("Yet Another Windows Dialer")
    msg = "Another Watcher is already running..."
ELSE
    WinTitle("", "Yet Another Windows Dialer")
    WHILE WinExist("Phone Dialer") == @TRUE
        WinWaitExist("Active call",1)
        IF WinItemizeEx("Active call",0,0) != "" THEN SendKeysTo("Active call", "~")
        IF WinItemizeEx("Warning",0,0) != "" THEN SendKeysTo("Warning", "~")
    ENDWHILE
    msg = "Goodbye!"
ENDIF
:theEnd
tme=TimeDate()
Display(5,"WinDial exiting at %TME%...",msg)
```

Considerations in Maintaining a Production Blaise CATI System

Kenneth P. Stulik, Total Service Solutions Inc. (for U.S. Census Bureau)

Introduction

The US Census Bureau's American Community Survey (ACS) is one of the largest continuous surveys in the world. The ACS is primarily a mail-out, mail-back survey which asks essentially the same questions as the Census Decennial Long Form. One part of the data collection effort is a Telephone Follow-Up (TFU) unit, which is essentially a CATI operation only for those mail returns that did not answer enough questions or had more people in the household than the form could accommodate.

The TFU CATI instrument is written in Blaise for Windows. What makes it different from most CATI instruments is that it must accept input data that represent the responses to questions in the mail questionnaire. This adds a layer of complexity since there are not only routines to maintain for periodic (daily, in this case) output of completed case data, but there are routines for the input of data as well. Control files exist to continuously assure that case counts between the various independent systems are synchronized. Reporting facilities gather information on daily caseload statistics, and automated mail routines notify a distribution list of either success or failure of the entire daily process.

Integrating and maintaining all of these different aspects of the system can be very involved. Developing routine maintenance, upgrading instruments or Blaise software, addressing problems, and optimizing the entire process require much time and effort. Efficient resolution of all of these different topics is paramount when processing the 150,000 cases that currently come to TFU annually. This paper is intended to assist the survey administrator who is facing the upcoming challenge of having to prepare or maintain the survey effort for such an initiative, and the paper will address the various challenges and solutions for handling common issues.

Operational Considerations

Automated Routine Maintenance

Perhaps the most important function of a CATI Administrator is simply the routine maintenance of the production environment – making sure everything runs smoothly for the interviewers and supervisors so that they can do their job. A simple way of accomplishing this is to add some sort of automated e-mail notification to your data processing routines, so that any number of people can know easily if there are problems to contend with before opening for business. We use the UNIX mailx command to perform this task. These reports can be as simple or complex as desired.

The purpose of these summary reports is two-fold. They not only inform all key personnel of the status of the system on any given day, but they also provide basic case counts: number of cases read in, number of cases read out, number of cases in the daybatch, number of cases in the control file, status of transferred files, and whether things are in sync with other control files. With this type of system, it is easy to tell the general status of your entire CATI production system at a glance.

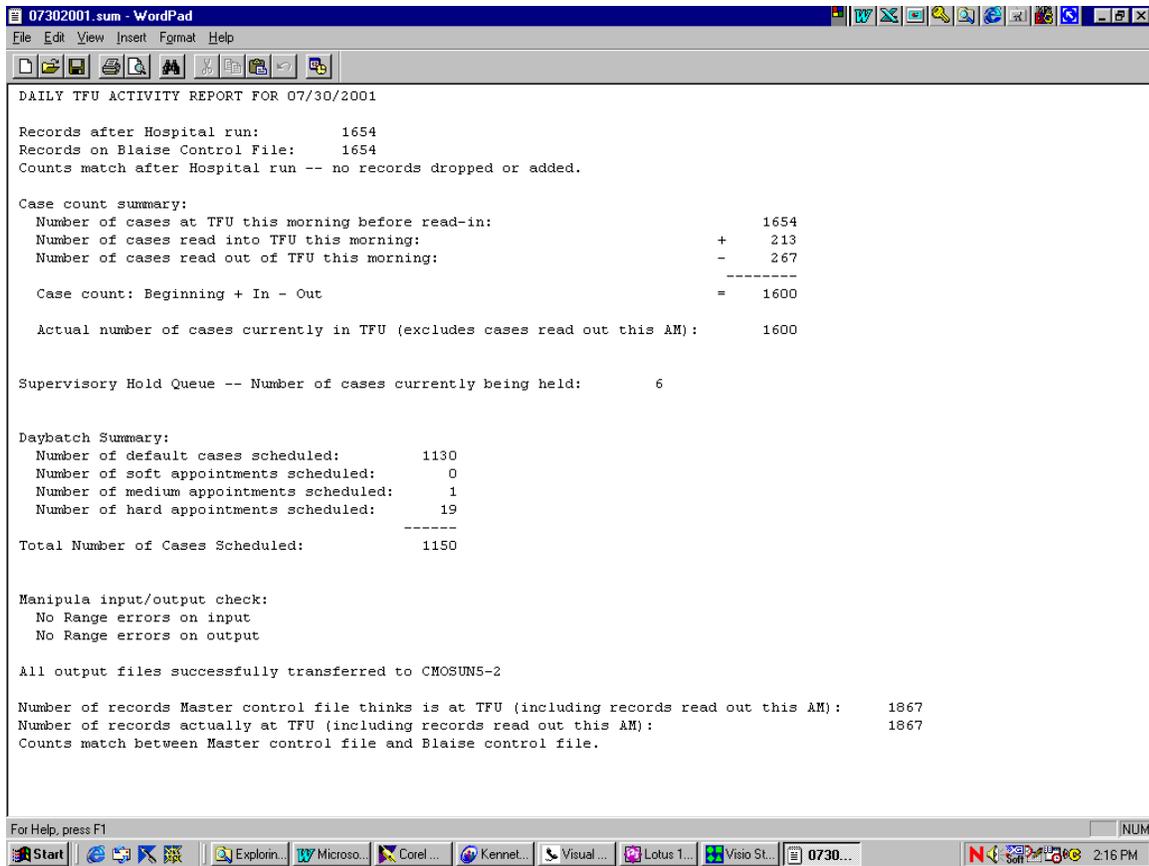


Figure 1. Screen shot of daily summary file.

Automated routines should perform a variety of functions beyond the read-in and read-out of data. Virtually all file archiving (covered in next section) can be handled in this way. Data integrity can and should be maintained in this process as well. The ACS automated routines perform a Blaise-to-Blaise Manipula transformation of the data and run Hospital daily. This aids in reducing Blaise database corruption, compresses Blaise database size, and optimizes primary & secondary keys. Reports of all kinds can be generated with these routines to keep track of any aspect of operational performance. Supplementary Manipula routines, daybatch creation, and file transfers also become part of this process. And detailed, dated logs of all of this activity make for a very robust application. Once fine-tuned and relatively error-free, this feature of the production system becomes the cornerstone of the CATI data processing operations and frees personnel up to do other work.

The ACS accomplishes this daily feat with a combination of tools. A WinAT task scheduler on a Windows NT Server launches a simple batch file every morning. The batch file in turn starts a SAS (third-party data management software suite complete with macro language and statistical functions) program which performs some of its own internal processing as well as the requisite calls to the OS to invoke Hospital and several Manipula routines. Flat files are transferred to a UNIX server which later looks for the presence of the deposited files and, if found, distributes them to a mail list.

How you configure your automated data processing is not so much the issue, so long as it is done. The process can range from simple to complex, and it can involve many sub-processes or few. It is best to use tools that you are comfortable with, tools that are well-supported within your organization, rather than tools that follow a given standard.

ACS Automated Processing as related to TFU

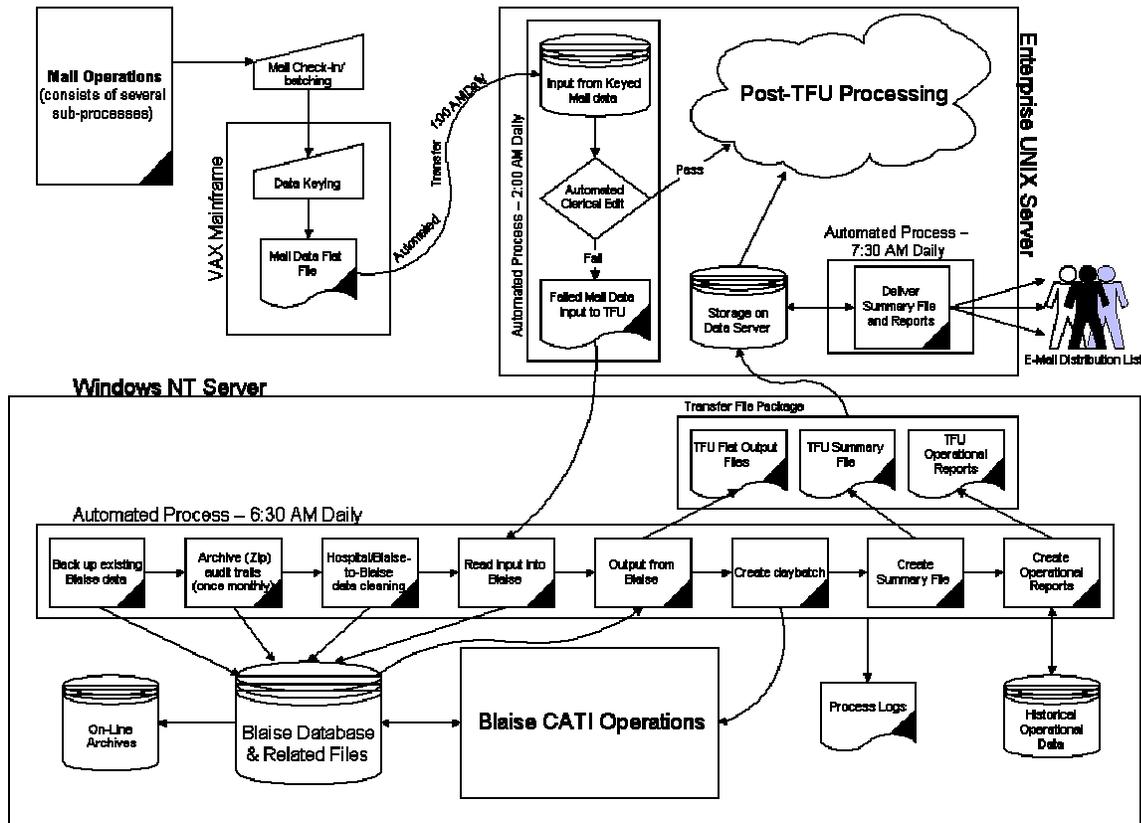


Figure 2:

Data-flow diagram of automated processing operations for ACS with regard to TFU.

File Maintenance

A natural side-effect of this type of automated production system is the continuous generation of many auxiliary files. One such file type is the audit trail. These useful files keep a text record of essentially every keystroke or navigational maneuver entered by an interviewer such that you can accurately determine what happened during any given interview. These files are extremely useful in troubleshooting problems with the Blaise Data Entry Program (DEP), since interviewers rarely recall the exact keystrokes they made in order to cause an error. The ACS TFU uses a modified audit trail dynamic link library which is designed to store audit trail info in text files named with the case's internal ID (also our primary key). This provides for easy access to a given audit trail, provided the case ID has been recorded or is able to be determined. But in the course of just one month, this can lead to more than 12,000 new files in your audit trail directory location! Most computers become very sluggish when asked to perform file operations on directories that contain a number of files of that magnitude, which can make the mere access of these files painstaking.

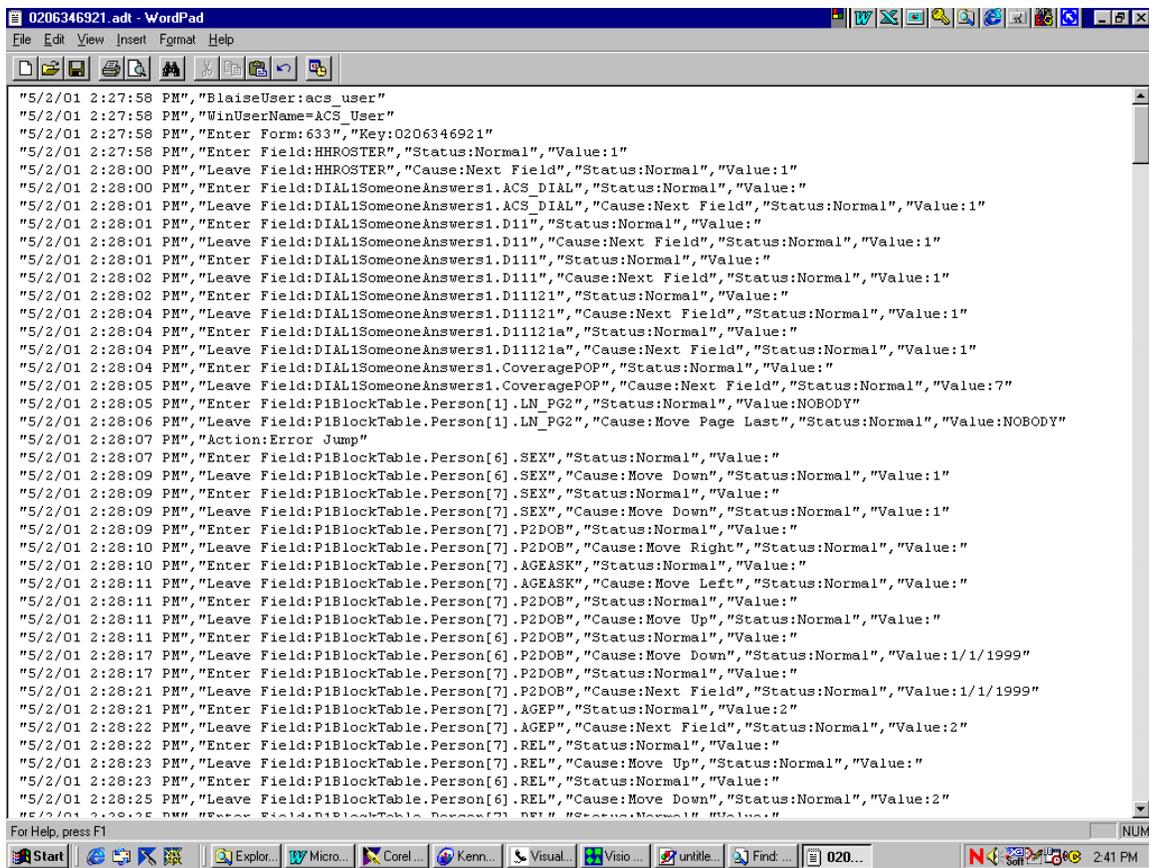


Figure 3: Screen shot of sample audit trail.

After a case has been removed from the Blaise CATI system, the need for an audit trail file is minimized. Such files could be deleted from the audit trail directory location with some pre-determined frequency, such as once per week, once per month, or whenever the case is read out of the system. But we have found that these files can come in handy even long after a case has been processed in the CATI system. Since they are simply text files, with detailed information as to the exact field and value that has been accessed, they can be used to reveal (or even play back) any given series of keystrokes, or even aggregated and scanned for various values in specific fields. This can lead to a crude but effective method for data recovery.

Archiving these files is a sensible and straightforward task to do, and it can be easily automated. Since they are text files, they compress very nicely (usually more than 90%) using a utility such as PKZip, or any compression software using the LZ adaptive dictionary-based algorithm. This compression can be automated to run at whatever time period you determine to be optimal to your production processes. Then you can store large numbers of audit trails in archive locations that take up only one file and are less than 10% of the total size of the archived files, and you can easily uncompress and access them later if needed.

Another type of file that can be archived in this type of production system is the flat input or output file. The ACS TFU receives input files and creates flat output files on a daily basis. Saving these files in archive directories is not only prudent, but easy to manage since there are only a few files per day. Although they can take up several hundred kilobytes or even a few megabytes each, they can be stored without compression and archived manually once every year or so.

Blaise data files themselves are a third type of file that must be managed regularly. In the ACS TFU production environment, these files are not saved permanently nor are they archived on hard drive for a long time. We will save three business days worth of Blaise data files, but after the third day, the data files are deleted. They can at that point only be recovered by tape archives, which go back two months. The reason for this is that not only are the Blaise data files, which include Blaise database, primary key, secondary key, join key, telephone history, log, and metadata files (to name the more sizable types) prohibitively large to keep stored on disk, sometimes reaching 100 megabytes or more in aggregate size, but the need for them is very low in a production environment. That isn't to say that there is no use having access to any given case in the Blaise database for any given day. It's just that if you have to reconstruct something from more than three business days prior, something we have found to be rare, that it is usually possible to do so by reloading flat files or recreating results by using audit trail information than it is to pluck out certain cases from a previous database and insert it into a current file.

Other files that you may choose to manage might be those file types that are ancillary to daily read-in/read-out operations such as message (*.msg) files from Manipula processing or any external files that may be necessary to provide for a historical reporting facility (see section on Reporting Tools). Again, these files are small in size (less than 100K each), and they number only a few per day, so keeping them stored un-archived in subdirectories is not a file or disk management problem. Also they can be useful for historical reference or troubleshooting.

Upgrades

No production system can be sustained for a lengthy period of time without upgrading some component. The Blaise system needs version upgrades because new desired features become available, or upgrades of new builds within versions are needed because of various software bugs. CATI instruments need upgrades because they also can be programmed with new features or to tweak that not-quite-correct procedure or include block. Automated data processing systems that supplement the Blaise CATI system need occasional bug fixes and enhancements. And the computer operating systems governing these various components occasionally need service pack updates or perhaps full migrations (e.g., Windows NT → Windows2000). I will address some of the issues involved every time an upgrade of any of these components occurs.

Blaise system upgrades

Upgrades to the Blaise system are occasionally necessary for new features or for bug fixes. Statistics Netherlands has been very busy with Blaise for Windows 4.x, offering no fewer than 6 distinct versions with the 4.x architecture and dozens of builds among all of the versions. Sometimes simply choosing which version to upgrade to can be an unclear choice. Consult with the experts at Westat or at Statistics Netherlands if you are not certain of your upgrade path.

Normally, you upgrade from your existing version to a higher build number within the existing version, or from a given version to the next higher version in sequence, say from 4.1x to 4.2. Build upgrades with versions are usually performed because there have been bug fixes or processing efficiencies added to the previous build for known issues. Otherwise, you can upgrade from one full version to another for the purpose of acquiring new features that can make your overall processes more functional, robust or efficient. An example of this would be upgrading from version 4.1x to version 4.2 for the purpose of taking advantage of the many new CATI call-scheduler features.

Blaise version upgrades are usually not particularly difficult. When upgrading within a version level, for example from version 4.11 to 4.12, or even when upgrading between versions 4.0x and 4.1x or between versions 4.2x and above, the actual upgrade is relatively straightforward. Installing the new version is as simple as archiving or deleting your old Blaise version, creating a new directory, and installing the new version to that directory. Sometimes the same license files are applicable. If a registry update is necessary (this would have applied to earlier versions of Blaise 4.x), a separate utility exists to do that.

Occasionally, there can be intricacies to Blaise version upgrades. For example, upgrading from version 4.1x to version 4.2 or higher involved the OEM to ANSI conversion of all code, metadata, database files, and so on. However, utilities and “wizards” for the transformation of all necessary Blaise files were provided, and with just some basic file management, making the upgrade and then transforming the files was not an overly complicated process. Future releases of Blaise versions will not likely involve a change of this nature, although other major changes could be involved. In any event, there will likely be a functional toolkit shipped with the product to facilitate whatever change is necessary.

The only real complication in performing a Blaise upgrade comes in managing multiple Blaise versions on your test environment. Almost always you will want to install the new version to test it, but keep the existing version around until the upgrade is complete. This has not been a problem for us, as we have found that multiple Blaise versions can exist harmoniously on the same server. When upgrading versions, we simply create a new directory and deposit the new version there. Any licensing information goes there as well, and if registry updates are needed we perform those as well. Then we point all of our shortcuts for Blaise programs in our test environment (such as DEP, Manipula, etc.) to the new version of Blaise, while keeping our copies of old production instruments and Blaise versions in their original places. Proper management of shortcuts and files allows for simultaneous testing of both old and new versions with little trouble.

The only caveat is that when upgrading to a new version that in turn updates the registry, we have found that all source code file become associated with the new version. So, for example, when invoking a current production piece of code, such as a .bla or .man file, you may expect the old version of Blaise to come up but instead the new version comes up but you do not notice. Then when you compile this piece of code, you may find that it does not work with your current production system. Careful attention to the particular version of Blaise under which you are compiling is prudent.

Instrument upgrades

Upgrades to the CATI instrument tend to occur when a bug in the source code has been discovered and the code is modified to correct the bug. But instrument upgrades can also be used to create a custom feature that may not yet be intrinsically available in the Blaise software. Regardless of the reason for the deployment, careful testing should always be done. This paper will not detail all of the steps involved in testing an instrument upgrade, although as a general rule, when a change of this nature is made, you should thoroughly check the DEP instrument, any input and output routines, and any auxiliary functions that rely on the Blaise metadata.

There are two main types of instrument upgrades: those that involve a structure change and those that do not. Those that do not are much easier to deploy. Since there is no change in database structure, no Manipula routines need to be recompiled, nor is there a need for any data transformation. The upgrade can be as simple as overwriting the existing metadata files with the new ones.

However, those upgrades that do require a structure change are very involved and prone to problems for two reasons. First, such upgrades in a production environment require that the Blaise database be mapped into the new structure. Second, virtually every Manipula program that uses new metadata model must be recompiled. If this should involve new input or output variables, that complicates the process even further, since the Manipula routines that perform read-in or read-out must be modified to accommodate the new variables. This additionally requires verification that the input/output routines are running correctly and a thorough examination of the database afterwards. While the actual amount of additional labor to do this type of upgrade is not extensive, the high possibility for error requires that you spend a significant amount of time testing before, during and after the actual upgrade.

Frequently, multiple bug fixes will be deployed in a given upgrade because in the time that it takes to change and test the new code, other bugs may be discovered. This is especially the case when we identify a bug or an decide to add a new feature that requires a structure change. Since we extensively test our new instruments when making even one small alteration that requires a structure change, it is efficient for us to deploy many bug fixes at once. We may sometimes many go months between instrument upgrades when a structure change is involved. However, for upgrades that do not involve structure changes, we will usually deploy those more rapidly and without extensive testing.

General considerations

We offer several considerations for testing an instrument upgrade, especially a major one in which a structure change is involved. First, it is highly advisable to have a separate testing environment, one which features a server of a similar type with similar operating system and having similar client stations as the actual production environment. For the ACS, we keep a complete, almost mirrored copy of the actual production environment on our test server. Under a slightly different root directory structure, there will be a test environment that has all of the new code to be deployed. In this area not only will subject matter testers give the new instrument a thorough examination with regard to the new structure, rules, CATI outcomes, textual content, and layout, but also all of the Manipula routines associated with the new metadata can be tested as well.

Second, just prior to conducting the upgrade, a dry run can be performed, all on the test server, in which the mock production environment on the test server is promoted to the new instrument and associated files. This not only serves as a test, but allows for practice as well. Furthermore, this methodology allows you to copy most compiled files from the test server to the production server when doing that actual upgrade, and this is usually much easier than recompiling everything at once.

Third, when upgrading, thorough and readily-accessible backups of all data, metadata, and source code files is essential. This allows for quick and easy recovery from any mishaps that may occur during the upgrade process. These backups may take up a considerable amount of space, but they need not be kept around indefinitely. As soon as the upgrade is complete and operational to the point that you are sure you will not need to recover from them, then these backup files can be deleted or archived to tape.

A fourth consideration when conducting a major upgrade is direct impact on interviewer down-time. Normally, you will not want to conduct an upgrade during normal business hours, as this will waste interviewing resources for an indefinite period of time. It is best to conduct major upgrades during off-hours. This gives you the flexibility to conduct the upgrade at a moderate pace, which should minimize the chance for errors, and it also allows for extra time if there are complications.

A fifth and final consideration when upgrading is the possibility of retrograding if necessary. If for any reason the upgrade cannot be carried out, say because of a significant bug that was not uncovered during testing or any other irreconcilable problem due to differences in the testing and production environments, you must be prepared to retrograde to the previous instrument or Blaise version. This is almost always undesirable, but it is better than having a system that is significantly deficient in some way. Retrograding is a very simple process if you have properly backed up all of the appropriate files.

Reporting Tools

A successful Blaise CATI operation must have a method for evaluating its performance on many levels. Reporting tools can provide insight as to how well the interviewing unit is staffed. Perhaps there are too many interviewers, or too few. Perhaps the overall number is correct, but too many during some periods and not enough during other periods. Reporting tools can also track individual performance for purposes of monitoring new interviewers to see that their productivity is at acceptable levels.

Reporting tools can also uncover group response rates, dial outcomes or any other call-level related statistic, and these can be broken down by number of dial attempts, time zone, hour of day or whatever other operational variable that is needed. This can be useful for determining things such as whether or not you are making enough or perhaps too many dial attempts per case, whether or not you are calling during optimal hours, or whether you are properly staffed and configured to handle time zone differences.

Reporting systems can be devised in a number of ways. Certainly, use of BtHist, Manipula, Maniplus, and Cameleon allow for extensive management of Blaise data. Developing reporting facilities with these tools is limited only by the extent of your organization's Blaise programming skill. However, third-party tools can be used as well. The ACS uses a combination of Manipula and the SAS statistical analysis system to manage the Manipula output and produce several canned frequency tables every day, week and month. The unit supervisors use BtHist during the business day to track operational progress by the hour.

An ideal tool for use that is currently in development within the ACS is the dynamic web-enabled report. Such a tool will theoretically allow any user anywhere within the organization's firewall to immediately access any report for any time period or to conduct real-time queries on aggregate, historical corporate data. The drawback to this kind of system is that it is not easy to construct. It requires programmers and designers who are capable of designing HTML or Java-based web applications and integrating them with the Blaise system, as well as systems administrators capable of configuring web servers and broker facilities that allow such web applications to operate.

Whatever method is selected is not of major importance, so long as some kind of operational analysis is being done. Without it, there can be no reliable way of tracking operational efficiency and progress.

Problem Resolution

One of the greatest challenges facing a survey administrator is handling problems that disrupt production activities. Some problems, such as power outages or crashed networks, are beyond the control of the administrator and must be tolerated. But most problems involve some kind of software issue, and when these problems arise, it is the responsibility of the administrator to effect a timely response.

The most threatening problems are those that bring the entire production system down indefinitely, or those that cause even a small amount of data loss. These problems require immediate action. There are several potential sources of errors that can bring production to a grinding halt: a Blaise system bug, a data corruption issue, an instrument bug, or even a network/OS/hardware problem.

One example of a crippling problem that involved data loss surfaced when we once did a major upgrade to our instrument. At the time, our read-in routines used a complicated ASCII-Relational method to process input data. Testing the read-in routines independently revealed no problems, but when we attempted to deploy in production, the input data did not correctly mesh with the existing Blaise database, resulting in loss of certain data fields upon input. When we tried to adjust the ASCII-Relational blocks upon read-in, that could not be successfully done. The solution was to immediately retrograde the instrument, redesign the read-in routines in a simplified ASCII method, test, and re-deploy.

Another example of a problem threatening production involves sections of data dropped by the Blaise system when BtMana failed to reassign a case's future status. Initially, these problems seemed relatively harmless, simply causing a case to receive default instead of the correct call scheduler treatment. But we soon discovered that chunks of data were missing from a few cases. We were able to correlate these cases with missing data to cases that had missing FutureStatus in the call scheduler. With some data samples from ACS, Statistics Netherlands was quickly able to determine that this was caused by an archaic byte inherent to our data model improperly interacting with a newer version of BtMana. Statistics Netherlands was able to immediately prepare a new build of Blaise for us, we quickly tested, and deployed the new Blaise version. We were able to use some Manipula routines to re-populate the cases with the missing data.

Regardless of the type or nature of the bug, the first step is to troubleshoot it as thoroughly as possible. Generally, issues with data corruption or missing data are related to the Blaise system somehow. We have found over the years that normally the instrument, operating system, network, or server is not responsible for errors of this kind. Issues which involve performance problems are usually tied to the design of the instrument, the overall configuration of the files in the survey, the network or the OS. Issues involving the system freezing between CATI calls usually are attributable to the Blaise system.

Sometimes administrators are faced with problems that make interviewing difficult or inefficient for the interviewers but do not significantly stop productivity. It seems that the majority of problems fall into this category. These can be tricky to resolve because an administrator must balance end-user needs, the feasibility of the timing of the solution, and the threat to data integrity based on whatever solution is chosen. For example, we encountered a seemingly frightening error using version 4.12 in which one interviewer's workstation would lock up, then all other interviewers would lock up when they attempted to retrieve the next case from the call scheduler. At first, this error seemed very serious and we assumed we would have to make an immediate correction. But as we investigated, we realized that: 1) the problem was very intermittent for us – once every few months, 2) the problem was easily remedied by finding the first offending client workstation and rebooting that workstation, and 3) the problem appeared to cause no data loss. Eventually the recommended solution to this problem proved to be to upgrade to Blaise version 4.5 build 640. But since we were not prepared for that deployment, and since the total amount of interviewer down time was about 15 minutes every couple of months, we decided not to rush the issue and wait until our upgrade was ready. This is a classic example of a situation in which the instinctive urge to immediately correct a problem would not have been the best choice.

Other problems may require diligence and patience to solve. At one point, our interviewing crew of 24 people began experiencing DEP performance problems, in which during certain periods there would be delays of up to 20 seconds between questions due to the instrument response time. This of course was a

major problem since a few respondents became very impatient and would abandon the interview. At first the problem was very difficult to pinpoint, because it coincided with other events on the Local Area Network (LAN) that would have been likely candidates to cause additional network traffic and produce wait states. But eventually these were found to be red herrings, as we discovered when using packet sniffers on the local router. Additional feedback from the interviewers helped us to pinpoint the exact times that the system slowed. Finally, we decided to conduct emulator testing on the CATI LAN. We found that when we ran a test on a separate but similar directory structure on the production server, we experienced similar delays in response time. But interviewers simultaneously in DEP in a different directory structure showed no slowdown. With the knowledge that the problem was instrument specific and occurred when a minimum threshold number of interviewers were active, we were then able to pinpoint the problem to our external lookup files. Placing those external files locally and reconfiguring the shortcuts proved to be the permanent solution.

When troubleshooting, use feedback from your interviewers, but be sure to verify and quantify things. Sometimes interviewers report certain conditions (e.g., “the computers are slow”), but don’t use relative terms, use quantifiable terms (e.g. “The response time is 2 seconds”). Interviewers can also inadvertently exaggerate, since perceptions can become distorted when handling hundreds of similar cases per day, so be sure to verify claims using audit trails and logs where possible. Also, be sure that reported problems are in fact unique and not just a minor variation on another problem that is already being addressed.

Expert help is available from Westat and ultimately from Statistics Netherlands. Know when to call them in. But be sure to have exhausted all standard troubleshooting procedures first. With the ACS having been one of the first continuous survey efforts to use Blaise in a unique mail follow-up mode as opposed to a traditional CATI mode, there were a number of bugs we encountered with DEP, Manipula, and data corruption. Both Westat and Statistics Netherlands were pivotal in giving us direction as to how to correctly diagnose the problem and in developing workarounds or interim builds to address our problems. Without their assistance, our continued progress in this effort would have been virtually impossible.

There are several steps you can take to minimize the chance that your system will be subject to problems:

- C Have the latest build of your Blaise version tested and in place in your production environment.
- C Assure that your Blaise instrument is coded as efficiently as possible, minimizing excess code, minimizing file access, and having ample comments in place to assist troubleshooting.
- C Make sure your server is optimized – run defragmentation regularly, scan for viruses regularly, have the latest operating system version (service pack) in place, run diagnostics intermittently.
- C Run Blaise data optimizing tools such as Hospital or a Manipula Blaise-to-Blaise transformation regularly, even daily, to minimize chance of data corruption and optimize keys.
- C Minimize clutter – archive or delete old logs, audit trails, Blaise databases, older metadata versions, or any other file that accumulates regularly and takes up significant storage space.

One other thing you can do to really minimize your problems is to set up your CATI environment on a separate server running within a separate LAN. We do not have the luxury of such a set-up, but the advantages in doing so are tremendous. One major advantage of such a configuration is that you can expect pretty consistent performance from your network. Another major advantage of such an isolated environment is that when troubleshooting, so long as you haven’t made changes to the server or the LAN, you can usually be sure that any problems you may have are not attributable to them.

Conclusions

When maintaining a continuous CATI survey operation, it is critical to have systems and procedures in place to handle both mundane and unusual situations as they should arise. An automated maintenance routine to process daily input and output data, generate reports, and alert personnel to problems is the centerpiece of any continuous survey effort. These features not only simplify day-to-day operational management for a small staff, but they allow optimization of the interviewing staff as well as data quality. Also, it is highly advisable to keep the complexity of such systems at a level that your organization is comfortable with. Having intricate systems and procedures for your survey administration may be appropriate, but they should also be within the scope of your ability to maintain and troubleshoot.

Established upgrade procedures, coupled with a well-structured test environment, are also a cornerstone of a successful survey effort. Without these, either the ability to upgrade software and instrumentation is compromised, or any upgrades performed run the risk of causing serious complications in the production environment. Thorough planning, testing, and documented procedures are a necessity for any administrator.

Quick and prudent problem resolution is another necessity. Most problems do not require immediate resolution, but all issues should be given an appropriate amount of attention under an overall plan to optimize performance of the unit – software, hardware, and personnel alike. Under such conditions, not only does data quality improve, but team morale increases as well.

By adhering to these practices, the overall functional quality of any continuous survey effort is significantly improved. The operation is more efficient; administrative staff members are more productive. Downtime is minimized, data quality is maximized. And ultimately, a smoothly running production system is well-suited for expansion.

Session 10: Experience and Organization

- Organisation of an Extensive Blaise Project
Holger Hagenguth, Federal Statistical Office, Germany
- Welfare Benchmark 'Easy to Use Local/Internet DEP' for the Dutch Municipal Government Officials
Carlo Vreugde, VNG, The Netherlands
- Blaise on Pentops, Blaise MetaData and Other Blaise Experience in the National Health and Nutrition Examination Survey
David Hill and Christina Kass, Westat, USA
Debra Reed-Gillette and Lewis Berman,
Centers for Disease Control and Prevention / National Center for Health Statistics, USA

Organisation of an Extensive BLAISE Project

Holger Hagengooth, Federal Statistical Office Germany

Table of contents:

1. Introduction
2. The German Microcensus
3. Goals of the reorganisation of the German Microcensus Blaise application
4. Folder structures
5. Data organisation
6. Pathnames in BLAISE
 - 6.1 Dynamic pathnames
 - 6.2 Static pathnames
 - 6.3 Sub-pathnames
 - 6.4 Pathnames in program development
 - 6.5 Pathnames when running the program
7. Working folder
8. Management of pathnames in a Blaise project
 - 8.1 Information on the static filenames in the commands USES, EXTERNAL, INPUTFILE, UPDATEFILE, OUTPUTFILE, LIBRARIES, RUN, CALL and EDIT
 - 8.2 Static file names in an INCLUDE file
 - 8.3 Dynamic pathname search using an MS-DOS procedure or a database
9. Data model
 - 9.1 Building up a complex data model
 - 9.2 Metadata model
10. Tools

1. Introduction

In the German official statistics, BLAISE is used, amongst other things, in two large, extensive surveys: the Microcensus and the ongoing sample surveys. These two surveys receive considerable attention from the political sphere, and the data gathered are requested and further processed by many social science institutes. BLAISE is therefore of great strategic significance to the German official statistics.

In May 2000, I assumed responsibility for the technical aspects of the German Blaise application for the Microcensus.

Until then, the BLAISE application had been developed by the specialist department, largely without support from the IT Department. As the years passed, the specialist requirements made of the BLAISE survey program became ever more stringent, and new possibilities were identified, whilst at the same time the procedures previously carried out on the mainframe were transferred into the BLAISE application. The German BLAISE application reached a level of complexity that could no longer be managed by the specialist department alone. The whole application had to be completely reorganised in line with modern, abstract principles of software development, including Internet- and object-orientated technologies, in order to retain its ability to be deployed flexibly and maintainably. This article explains how these principles are implemented in BLAISE, and which concepts of BLAISE can be used for this. My many years of experience in managing complex EDV projects in the German official statistics were very helpful here. Since there was very little knowledge of many of these possibilities among German BLAISE programmers, it appears to be very necessary to point this out in particular. Many of the ideas put forward in this article are of value in my view for all the more complex BLAISE applications. Guidelines should be developed for other German BLAISE applications on the basis of this paper: for example, to ensure that all applications in future use modern principles and that the software development is organised effectively, standard modules can be developed allowing the wide variety of features offered by BLAISE to be put to better use in the German official statistics.

In Chapter 2, I will report on the organisation of the German Microcensus and the resulting requirements made of the BLAISE program, and in Chapter 3, I will introduce the desired aims and principles of modern software development, then in the following Chapters, I will describe how these goals can be implemented in BLAISE.

2. The German Microcensus

Roughly one percent of the German population is surveyed once every year in the German Microcensus statistics (by data on identity, family connections, economic and social situation, training, occupation, residence, health, etc.). The selection of the surveyed population is effected by a mathematical random sample procedure.

The list of questions changes each year, several new questions being added and several being deleted, whilst some old questions are re-activated.

The German official statistics are organised in line with the principles of Federalism. The actual survey of the data is effected in the Federal Länder and the local Land Statistical Offices. The Land Statistical Offices provide the data to the Federal Statistical Office, where they are combined. In accordance with federalist principles, the Federal Statistical Office primarily has the role of coordinator for the State Offices. The technical equipment of the individual State Offices varies greatly, and the organisation of work within the individual authorities is based on a variety of structures. In order to ensure that the data in Germany can be combined and surveyed in an uniform manner, the same BLAISE program is used to collate the data in all State Offices. This BLAISE program is developed centrally at the Federal Statistical Office, and then provided to the 16 Land Statistical Offices, where it is installed and used. No programming is carried out in BLAISE in the 16 State Offices, so that only prepared objects are provided, and no sources. The various sets of technical and organisational conditions at the Land Statistical Offices must be taken into account in developing the application at the Federal Office.

The population is surveyed by interviewers who visit the households in person to collect the data either with a laptop or a paper questionnaire. Data are also transferred from the laptops to the Land Statistical Offices in very different ways across the Federal Länder (modem or diskette, daily or at the end of the survey). The data collected using paper questionnaires are integrated into the BLAISE program in the Land Statistical Offices.

These points lead to the following requirements for the BLAISE program:

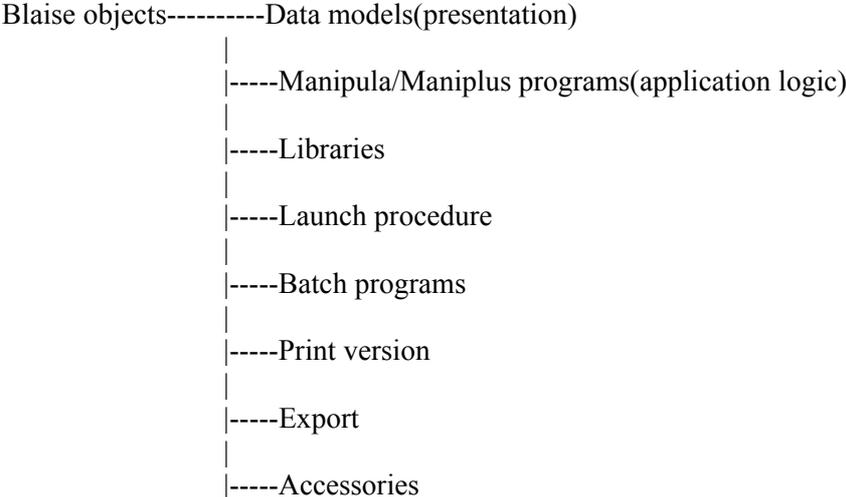
1. Flexibly controlled data model (list of questions)
2. The program should be deployable in and adaptable to a wide variety of technical and organisational structures.
3. The program should be adaptable to various environments by environment variables and a profile data model.
4. Installation of the application should be simple and flexible, largely without specific requirements.
5. It should be possible to distribute the application in various network environments.
6. Data backup and transfer largely independent of technical requirements.

3. Goals of the reorganisation of the German Microcensus Blaise application

It should be possible to maintain a strict separation of the Blaise objects and the data objects within the application, so that several versions can be processed, test data can be prepared, and learning data can be set up, etc.

As stated above, the names "Folder at the installing Office Version1, Folder at the installing Office VersionN" can be selected at will.

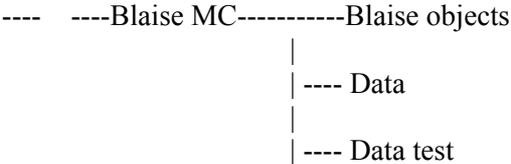
In order to clarify the structure of the application, the Blaise objects folder has been further subdivided:



In an organisation such as the German official statistics, guidelines and rules should be defined for this subdivision to enable rapid orientation in all Blaise applications within the organisation. Thus, standard modules can be developed and synergy effects can be used.

5. Data organisation

The strict separation of data and Blaise objects was pointed out in item 3 above. In a Blaise data model, much metainformation, such as information on checks carried out, was stored in addition to the data. The effect of any change to the CHECK rules is that the old data become unreadable. This fact has been frequently criticised within official German statistics. This problem can however be solved by sensibly organising the data and the check data:



In the "Check data" folder, there is for each data model of the "Data" folder a data model containing only the fields of the data model, but not the check rules. If a change is made to the model in the "Data" folder, the data from the "Data" folder can be transferred to the "Check data" folder at any time. Once the change has been made, the data can be read back. If a field is changed in the "Data" folder, this must also be entered in the "Check data" folder. The "Check data" folder therefore serves as abstract data storage. Corresponding fields or notes can be used to enter the significance of the check data record and then searched for, functions that are not required in the "Data" folder. It is also possible to create the "Check data" folder as a database independently of Blaise (for example using ACCESS). The principle of saving and reading back works in the same way.

6. Pathnames in BLAISE

The pathnames and filenames in Blaise must be given particular attention so that the abovementioned goals can be achieved. Each object called up in Blaise must be fully identifiable in the network environment; the complete pathnames and filenames must be available.

With the need for flexible installation of the Blaise application, and installation of the application using any chosen folder name, the programmer does not know the pathname, so it must be determined when running the program. The full pathname must be put together implicitly or explicitly in the Blaise program when it is running.

6.1 Dynamic pathnames

All pathnames that the programmer does not know, and all pathnames dynamically determined either implicitly or explicitly when running the application, are referred to below as "dynamic pathnames".

If the application is installed using the following structure:

```
----  ----- Folder at the installing Office Version1-----Blaise objects
                                         |
                                         |--Data
```

the pathname of the drive up to "Folder at the installing Office Version1" is a dynamic pathname.

6.2 Static pathnames

```
Blaise objects-----Data models(presentation)
      |
      |-----Manipula/Maniplus programs(application logic)
              |
              |-----General programs
                      |
                      |-----Main program.msu(msx)
```

The path "Blaise objects\ Manipula/Maniplus programs (application logic) \ General programs" is known to the programmer who develops the project because he/she decides on the name.

6.3 Sub-pathnames

Consider the following example:

```
C:\Ord1\Blaise\Blaise Microcensus\Blaise objects\Manipula/Maniplus Programs\General
programs\Main program.msu
```

The complete pathname is: "C:\Ord1\Blaise\Blaise Microcensus\Blaise objects\Manipula/Maniplus programs\General programs", whereas "C:\Ord1\Blaise\Blaise Microcensus" and "Blaise objects\Manipula/Maniplus-programs\General programs" are two sub-pathnames of the complete pathname. By combining (concatenating) the two pathnames, the complete pathname is created. In most cases, the complete pathname consists of a dynamic sub-pathname and a static sub-pathname. How the pathnames are divided into sub-pathnames, which of them are dynamic and which are static, which operations are defined for pathnames, is the basic project decision which determines the flexibility of the installation.

6.4 Pathnames in program development

If the source of a Blaise object is loaded into the text editor of the Blaise Control Center, the intention is to be able to implement the Prepare command directly. For this, all data models needed for conversion (USES section) must be found (i.e. with the full path). In the source code of the object, therefore, only the static sub-path should be stated. The dynamic sub-path should be permanently set in the project options. This means that the dynamic path for running the project must also be entered statically for development. Since the development takes place in a fixed environment, this is not so laborious; it only has to be entered once in the projects, and then applies to all objects within the project. If a new version of the project is also created for development, then in the new version only the pathnames in the project objects must be adapted; this is not a burdensome requirement. If the B4Cpars.exe interface is used to prepare objects (for instance using a procedure) all dynamic sub-pathnames can be determined while the application is running and transferred as parameters. Therefore, the static entries in the options of the project are not needed. This is however not practical during development. A new version of the project is not needed that often.

6.5 Pathnames when running the program

All static sub-pathnames in the source of a Blaise object (USES, EXTERNAL, INPUTFILE, UPDATEFILE, OUTPUTFILE, LIBRARIES) that are permanently allocated can be overwritten while the application is running, and the commands CALL, EDIT and RUN can be transferred as parameters. The parameters are described in the Developers Guide on pp. 667 et seqq. Thus, the folder structure can be made flatter in the "Blaise objects" folder than in the "Sources" folder. When the application is running, a description of the structure may not be required in the same form as it is during development of the application in the "Sources" folder. Also, the division of the complete pathname into dynamic and static pathnames can be effected differently while the application is running than during project development.

7. Working folder

The working folder is the folder accessed by the application with MANIPULA.EXE. The working folder forms the basis for DOS commands made in the Manipula program, and sub-pathnames stated in the USES INPUTFILE refer to the working folder. The treatment and administration of the working folder is hence of fundamental significance for the organisation of the pathnames.

It is possible to determine the working folder during running by the following means:

- to use the MS-DOS procedure and environment variables, which cannot entirely be done directly,
- in the Manipula start program to use a dummy file and the Manipula/Rules function pathname. This dummy file must not have any path information in the INPUTFILE section, and must be located in the same folder as the Start program.

It is possible to change the working folder during running by the following means:

- To use the Run and CD commands in a Manipulus program e.g.: Reslt := RUN ('CD STARTPROCEDURE' , WAIT)
- To use the parameter /W when calling up Manipula.exe, Dep.exe

If the working folder was set when launching the application in the Start program, it can be transferred using the parameter /P to all Manipulus programs and it is available there. If the Start program launches a Manipulus program in another folder, such as: Reslt := CALL('Manipula-Maniplus Programs\General_Programs\ AnzPC), the working folder of the start program remains the present working folder in the AnzPC program. If the AnzPC folder is intended to become the present working folder, /WManipula-Maniplus-Programs\General_Programs must be transferred as a parameter.

Main program:

USES

```
look      'Data models\Organisational data\Look\Look'  
doublecd 'Data models\Organisational data\doublecd\Doublecd'  
staff     'Data models\Organisational data\Staff'  
MaxP      'Data models\Microcensus data records\MaxP'  
err_lap   'Data models\Organisational data\err\Err_lap'
```

```
INPUTFILE  colleg: staff('Data\Organisational data\Staff',BLAISE) SETTINGS OPEN=YES  
ACCESS=SHARED
```

```
INPUTFILE  looknb: look ('Data\Organisational data\Look\Look',BLAISE)  
SETTINGS OPEN=NO
```

The effect of the CALL command ('Manipula/Maniplus programs\General programs\Main program') is that the name Manipula/Maniplus programs\General programs\Main program is implicitly supplemented dynamically during running by the working folder, and is correspondingly found and launched. The same applies to the data models and files. All names apart from the working folder are therefore static and stated permanently in the source code.

Advantages: maximum performance, very easy to understand, simple to install

Disadvantages: lack of flexibility, no way of distributing the application on a variety of drives, no differentiation between program development and running, and hard to change: if the structure is expanded or changed, all changes must be entered in the sources.

8.2 Static file names in an INCLUDE file

When a Maniplus setup is called up, the name of the chosen setup does not have to be explicitly stated; a STRING variable is possible.

All static pathnames/filenames can be listed in an INCLUDE file in string variables; the system setup can be highly flexible with a corresponding name convention for the string variables.

Definition of the string variables:

AUXFIELDS

{The variables for the filenames and pathnames are defined below.

Only relative pathnames are needed here for data files and MANIPULA/MANIPLUS SETUPS.

All other relative pathnames are stated in the program.

The following convention is introduced here:

String : This is the character chain of the name

Path : This is a (relative) pathname

File : This is a (relative) filename

from : The path (filename) starts with the folder following 'from' (drive)

to : The path (filename) ends at the folder following 'to' (filename)

The variables are set at the start of the program (in the 2nd INCLUDE file) so that in the event of a change to the path structure only this variable setting needs to be changed.

System refers to the folder in which the BLAISE application was installed.

If the variables with + are concatenated, it is not possible to recognise immediately which path is formed. A pathname starts with a letter and ends with a \ .}

```

String_Path_from_Drive_to_Blaise_Objects_to_Run      : STRING [255]
String_Path_from_Drive_to_Data                       : STRING [255]
String_Path_from_Data_to_Data                       : STRING [255]
String_Path_from_Classifications_to_Classifications : STRING [255]
String_Path_from_Guide tapes_to_Guide tapes         : STRING [255]
String_Path_from_Organisational data_to_Organisational data : STRING [255]
String_Path_from_Organisational data_to_Save        : STRING [255]
String_Path_from_Organisational data_to_Copy        : STRING [255]
String_Path_from_Organisational data_to_Count       : STRING [255]
String_Path_from_Microcensus data records_to_Microcensus data records : STRING [255]
String_Path_from_Microcensus data records_Laptop_to_Microcensus data records_Laptop :
                                                    STRING [255]
String_file_from_Manipula_Maniplus_programs_to_Start_MC3_Check : STRING [255]
String_file_from_Manipula_Maniplus_programs_to_Main program : STRING [255]
String_file_from_Manipula_Maniplus_programs_to_dataretg : STRING [255]
String_file_from_Organisational data_to_Staff       : STRING [255]
String_file_from_Data models_to_Staff               : STRING [255]

```

Occupation of the string variables:

```

String_Path_from_Drive_to_Blaise_objects_to_Run      := PARAMETER(1)
String_Path_from_Drive_to_Data                       := PARAMETER(2)
String_Path_from_Data_to_Data                       := PARAMETER(3)
String_file_from_Manipula_Maniplus_programs_to_Main program :=
    'Manipula-Maniplus programs\General_programs\Main program'
String_file_from_Manipula_Maniplus_programs_to_dataretg :=
    'Manipula-Maniplus programs\General_programs\dataretg'
String_file_from_Organisational data_to_User         :=
    'Organisational data\User'
String_file_from_Organisational data_to_Staff        :=
    'Organisational data\Staff'
String_file_from_data models_to_Staff                :=
    'Data models\Organisational data\Staff'
String_file_from_Organisational data_to_pcnun       :=
    'Organisational data\pcnun'

```

Accordingly, variables can be defined for calling up a SETUP:

```
String_Call_up_Main program : STRING[255]
```

And this call up variable is allocated according to the parameter conventions: String_Call up_Main program :=

```
String_File_from_Manipula_Maniplus_Programs_to_Main program + '/W' +
```

```
String_Path_from_Drive_to_Blaise_objects_to_Run + ... .
```

The call up is then: CALL (String_Call_up_Main program)

Advantages: Separation of development and program running, easier to change since in the event of a change, only the entry in the String variable must be effected, and not in each separate object. No measurable performance disadvantages as against 8.4. Easy to install.

Disadvantages: Since the INCLUDE file is read when preparing the object and is permanently anchored in the prepared Blaise object, a flexible distribution of the application is not possible on installation. Somewhat more work is involved in programming.

8.3 Dynamic pathname search using an MS-DOS procedure or a database

As in 8.2, all call ups from Maniplus setups and links to external files are implemented via variables. These variables are set dynamically when running. There are very many ways of doing this. Basically, the system can be designed to be as variable as possible:

- Each object is given a name (a virtual address). Using this name, the current filename and pathname is found in a database. Hence the application can be installed and distributed with considerable flexibility. If the application is distributed differently, only the entries in the database need to be changed, and the application can run. In the same way, the external objects needed by a program are entered, and the interface is built up dynamically in line with the name (using a separate interface file for each object).
- The object is found in selected directories, for instance by an MS – DOS procedure, and the complete pathname and filename is set in this manner. The same procedure can be followed with the required external objects for calling up the program.
- The system administrator can use a profile data model to enter drives and pathnames.

Advantages: Separation of development and program running, easy to change, highly flexible in distribution and installation.

Disadvantage: More burdensome to administrate, higher performance, harder to install.

Since this is a procedure that does not depend on projects, the procedures that have been developed (modules) can be provided for all projects. The greater effort needs to be made only once.

9. Data model

Various procedures can be used when building up a Blaise data model. It is possible to program following the progression of the questionnaire, such as:

```
IF EF1 = 1 AND EF2 = 1
  THEN
    EF3
    EF4
  ELSE
    EF5
    EF6
ENDIF
```

This leads very quickly to conditional constructions that are hard to follow.

Since the display of the fields and their sequence is set statically in the electronic questionnaire, a field-related view is recommended:

```

IF EF1 = 1 AND EF2 = 1    { enquiry from EF3 }
  THEN
    EF3
ENDIF
IF EF1 = 1 AND EF2 = 1    { enquiry from EF4 }
  THEN
    EF4
ENDIF
IF NOT(EF1= 1 AND EF2 = 1)    { enquiry from EF5 }
  THEN
    EF5
ENDIF
IF NOT(EF1 = 1 AND EF2 = 1)    { enquiry from EF6 }
  THEN
    EF6
ENDIF

```

For each field, the condition is given in its entirety as to when it is queried, when it is displayed, when it is possibly deleted, etc.

The core of this method of observation lies in the field and not the enquiry logic. The application is given a clearer structure. After each field enquiry, the checks are performed. The checking logic is unlinked from the enquiry logic as far as possible. The core of the check logic is a complete test. Each test is effected using its own check routine in which all fields that are important in the check are used as parameters.

Below, therefore, the method is described that is used in developing the highly complex German data model for the Microcensus.

9.1 Building up a complex data model

- Determining the field sequence to follow a possibly existing paper questionnaire
- It is determined for each field when it is queried, when it is displayed, etc.
- For each field, the plausibility checks are described and programmed in a separate routine referring to this field and to all previously enquired fields.

Program example: enquiry logic:

{Enquiry of the Change field}

```

IF Result = Implem AND HHNumber <> 00 OR
  Result = questioned
  THEN
    IF RedV = 1 OR RedV = 2 OR RedV = 4
      THEN
        Change
      ENDF
    ENDF
  ENDF

```

{The following checks refer to the fields:
Result ResultVJ HHNumber HHNumberVJ Redv Change}

Check_BG09 (HHNumber, RedV, Result)

{N.B.:
The result 'GoneDead' may not apply to a household newly included in the survey.}

Pruef_BG10b (RotV, HHNumber, HHNumberVJ, ResultVJ, Result)

{N.B.:
In 2000, the household had the questionnaire result '^ResultVJ' (gonedead).
Therefore there cannot be a result to the questionnaire for 2001.}

Check_BG10a (RedV, HHNumber, HHNumberVJ, ResultVJ, Result)

{N.B.:
A household number used the previous year for an empty dwelling can only be
used for the dwelling if it is still unoccupied.}

Check_BG12 (RedV, HHNumber, HHNumberVJ, ResultVJ, Result, Change)

{N.B.:
The new inclusion of the household was accepted although the questionnaire result in
2000 was "questioned" or "cancelled".}

Program example: Check routine

{Check BG04}

PROCEDURE CHECK_BG04

PARAMETERS

PHHNumber : THHNumber

PResult : TResult

PNo.Pers : TNo.Pers

PKomp2 : TKomp2

RULES

IF PResult = Implem AND PHHNumber >= 01 AND PHHNumber <= 98

THEN

IF PNo.Pers <> DONTKNOW

THEN

PKomp2 = DeuOA OR PKomp2 = DeuHw OR PKomp2 = DeuNw OR PKomp2 =
Ausl OR

PKomp2 = StAoA

"This category cannot be correct since the household size

(^PNo.Pers person(s)) is stated!@/

If no further information is available, category '5'

(nationality of reference person unknown) is to be stated.(BG04)"

ENDIF

ENDIF

ENDPROCEDURE

9.2 Metadata model

If the data model is structured as in 9.1, it is very simple to control the data model and, for instance, to activate and deactivate checks and blocks, etc. It is much more difficult, if not actually impossible, to control enquiry logic.

For the data model, a metadata model is formed, containing a field for each block and each check, which is used as a switch determining whether the enquiry or check is to be implemented:

```

Check:
IF P_F_Aus.S_P_Check_No.pers = Yes
    THEN
        Check_No.pers (No.pers)
ENDIF

```

Enquiry:

```

IF P_F_Aus.S_B_FNR = Yes
    THEN
        TFNR
ENDIF

```

10. Tools

If the Blaise application is developed using the following structure:

```

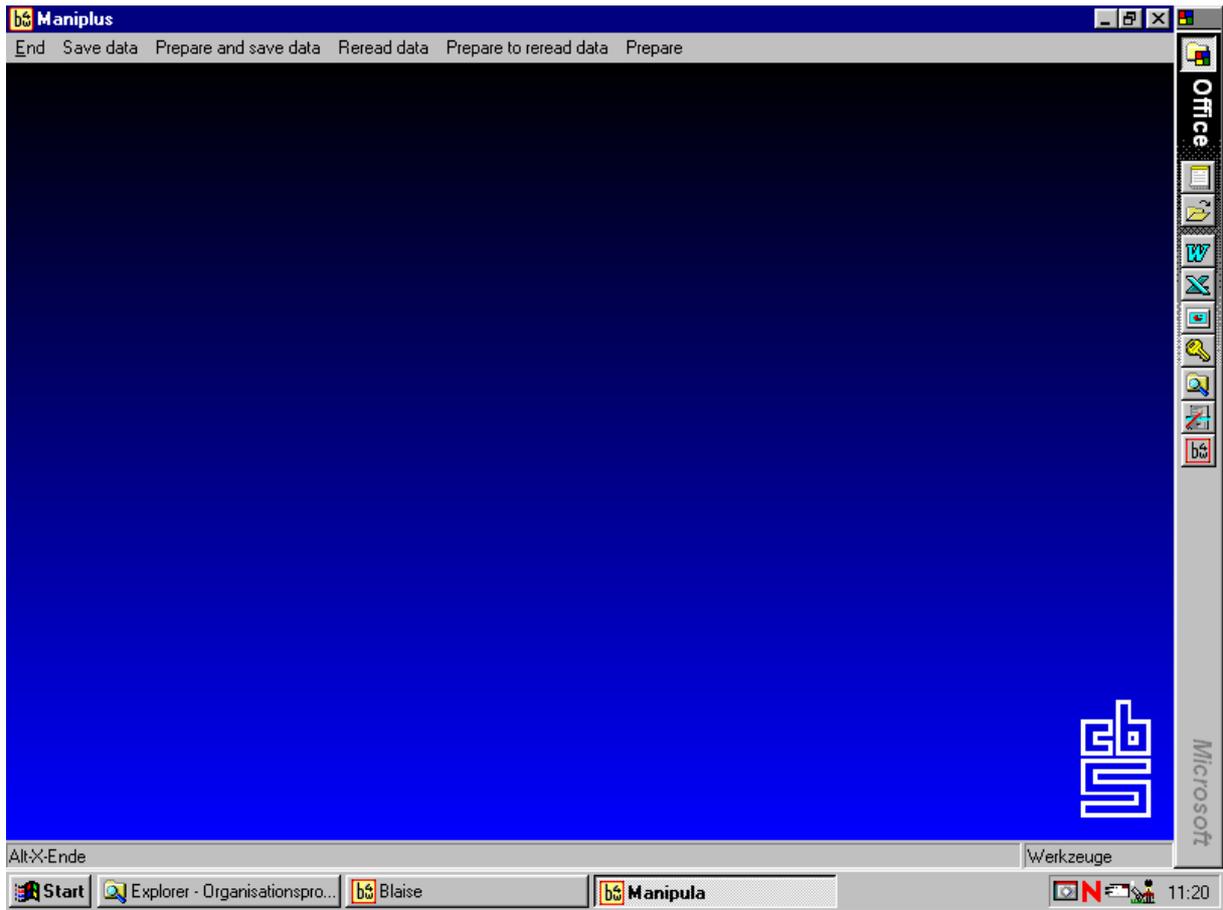
----  ----Blaise application-----Blaise objects
                                     |
                                     |----Data
                                     |
                                     |----Data test
                                     |
                                     |----Sources

```

it is possible to create extremely effective, powerful tools with which the development can be greatly accelerated.

- A tool to prepare all objects in a batch.
- A tool to prepare selected objects, such as all Blaise source files containing a text string. If a data model is changed, all Maniplus objects in the USES section needing this data model are found and have to be newly prepared.
- If the "Data check" folder is organised as an abstract data store – cf. 5. – tools can be formed to store and re-read the test data. If a check rule is changed in a data model, the data are transferred after a data check. The data model is changed; the data are re-read and are available once more.
- Tools to manage the project automatically, for example to compare the "Sources" and "Blaise-objects-to-run" folders. If for each source an object in exists in "Blaise-objects-to-run", and vice versa, the versions are correct.

All these tools can be launched using a Maniplus application which is available and can be used constantly during development:



Welfare Benchmark ‘easy to use local/internet DEP’ for the Dutch Municipal Government Officials

Carlo J.C. Vreugde, VNG, SGBO, StimulansZ, of the Netherlands

1. Introduction

The Welfare Benchmark of the Netherlands is a project for the Dutch Municipalities to learn and improve their method of applying Municipal welfare programs. The Welfare Benchmark collects data from the Dutch Municipalities using Blaise DEP programs to form a unique central database. The big challenge in this National project is to use Blaise at its maximum potential: easy data-entry for non Blaise-users, namely Municipal-government-officials.

It is the development and implementation of an easy to use Blaise DEP program and all the procedures of creating one central database that this paper addresses. We were confronted with the following technical aspects: easy DEP installations, easy DEP use, easy returning of DEP data and easy DEP data editing; all done locally and through the internet.

Finally, this paper will address which important elements made the Welfare Benchmark so successful and the working relationship with the Central Bureau of Statistics Netherlands and CenterData which contributed to its successful implementation.

2. The target group

The VNG is the national organization for Dutch municipalities. It contains the nonprofit agency StimulansZ that advises all the municipalities about handling the welfare grants from the government for its welfare recipients. With the help of SGBO(research and consultancy department of the Dutch Local Government Association,VNG) and CentERdata (research expert center of the University of Tilburg) it has made a special easy to use Blaise Data-entry program for the Dutch municipalities.

Dutch municipalities send their government official, who is responsible for the Welfare program, to a major meeting of StimulansZ. In large municipalities, this person will delegate the Benchmark project to other government officials but in regular municipalities they would have to use a very easy and uncomplicated data-entry program. Otherwise the Welfare Benchmark would be a total failure.

Last year another program than Blaise was used which caused a lot of problems (Qubus). It was also supposed to be easy but the program accepted data-entry errors and therefore the final database became very unreliable. The Blaise DEP program had to be easy to use but without data-entry errors.

The other program also had a lot of installation problems. This was because all of the Dutch Municipalities have MS Windows but all the system operators made different environments, for example they closed off certain tools and functions like email and major document extensions. Therefore Blaise DEP has to be very easy to install in any MS Windows environment.

Our major task was not to scare off our target group, by using Blaise DEP, and at the same time get the best results possible.

3. The easy installation of Blaise DEP

Blaise Data-entry program is used by the StimulansZ Welfare Benchmark to collect data from a 100 volunteer municipalities. Through this data, StimulansZ will measure long-term progress in achieving the Government strategic plan goals: municipal cooperation, collaboration, and a result-driven approach to problem solving. The results will be compared by how compatible municipalities as a whole are doing, and by performance measurement systems, which seek to measure how a given municipality is performing. Municipalities are therefore clustered in groups of 8 who have similar backgrounds.

“The comparison circle method, as well as all other benchmarking methods, is a quality management instrument. It originates from the desire of local government organisations to be able to compare the individual organisation, services and policy, with peer organisations, in order to improve its quality of services and products. Important in the work with comparison circles, is the fact that the initiative lies with the local authority organisations and is not forced onto them in a top down way by another agency. It is the story behind the data that counts. Comparing the information makes the participants want to look for and discuss the reasons of the differences.”(Drs. Piet C.A. Severijnen Senior Researcher/ Consultant SGB0-VNG, Benchmarking for local authorities according to the comparison circles methodology.)

The best way to reach the municipalities in the Netherlands was through email, or so we thought. Make an individual Blaise DEP program as small as possible and use email to send the questionnaire. It was not as straightforward as we had assumed.

A mailing with questions about the municipal email facilities was conducted among all the participants in the Benchmark. A total of 110 municipalities replied and out of that total only 25 had email restrictions: they could not receive email or they could not receive email attachments larger than 1 megabyte. During the pilot we emailed *.zip files which could not be unzipped by some of the pilot-municipalities. Therefore we made self-extractable *.zip files into the *.exe files. This was a disputable decision, because municipalities who could receive email larger than 1 megabyte could not receive attachments with the extension *.exe due through email-securities. In the end we had to send 50 Cd-roms with 1.5 Megabytes.

3.1. The easy use of Blaise DEP

How would you email a Blaise DEP program which would be easy to install on any computer? We thought of using a self-extractable zipfile which would make the use of Blaise simple and understandable for everyone.

The zipfile would unzip at the C:\ drive (with permission) and make a directory C:\BENCH , in there were two old dos *.Bat files: start.bat (wich would activate Blaise DEP with the municipal-unique number [DEP.EXE /G /K14 bsd] and return.bat (wich would zip all the files starting with a B*.*) and lastly another directory C\BENCH\MARK*.* with all the Blaise files.

The government official had to make MS Windows shortcuts from the start.bat and the return.bat to his MS Windows Desk. With one click on the start.bat shortcut Blaise would activate. When all the data was entered he could click on the return.bat shortcut and everything would be placed in return-zipfile in the directory C:\BENCH with the municipal-unique number. [This was done by ZIPDLL.dll which was made by Bas Weerman of CentERdata] Using his own email program he had to attach the return-zipfile and email this to our organization. In our Pilot this was the function that we improved a lot. In our initial idea we only emailed the Blaise database file back, *.bdb. This was not sufficient because we were using the remark ‘paperclip’ function of Blaise. This functionality creates and uses more files in the Blaise *.bdb directory. Therefore all the files with the programname bsd*.* were zipped and returned to us by email.

All it took to work was to make two MS Windows Shortcuts and a copy of one file in an email program. From the 100 municipalities we received no telephone calls for these three actions.

The next step was the Blaise DEP program itself. A government official was becoming a data-entry person. He was not familiar with any data-entry program or activities in this field.

Therefore we made the Blaise DEP program super simple.

The government official is welcomed by the name of his municipality on the first page

The first page also tells him there are 5 chapters, and to click on one of the Tab-sheets

By clicking on a Tab-sheet at the top he will see all questions of one chapter

Chapter shows the Question and information and the answers on the Left-side

All the question-numbers –enter boxes are displayed on the Right-side

The exception is the question-tables which are on separate pages

There are only 6 speed buttons: Don't Know, Remark, plus 4 navigate buttons

With the same 6 menu-items as in the speed buttons

With an auto save-interval of 3 minutes and auto-save when finished, all goes automatically, no worries about data saving.

At the end of each Tab-sheet are two questions: 1. Would you like to print your entered data of this chapter? 2. Would you like to continue to the next chapter after printing?

The most important part was to make it look simple and finally to enter data easily. Just thinking about entering 135 questions is a big task for a government official. To enter data correctly is in the best interest of the government official so the first look at Blaise DEP was very important.

The DEP menu file controls the menu and speed buttons available in the DEP. The first step was to make the menu-items and speed buttons. We started out with only 2 speed buttons but we found out that navigation through the Blaise DEP became more important when the government official had already entered most of the data. Next, we made the menu-items identical to the speed buttons. This was also to translate everything to Dutch in the DEP menu file.

We had to use the Modelib-editor very thoroughly. Use of easy colors and layout were very important; off-white for the questions and off-green for the answers. The DEP should look and feel comfortable to the Government official. In the options we activated the 'Show parallels on tab sheets'. This activates the easy to use tab sheets in the whole DEP program. The next important feature was the activation of the Field description in Field Panes of Layout Interviewing. This meant that we had to use Field descriptions in the *.bla. At the end of every field [..." / "9" :T9] we placed the question number. We also used three columns in the Grids, Interviewing, and Layout. Through this method we created two major areas: one left side with info panes and on the right side a three-column grid.

3.2. The *.bla for the easy use of Blaise DEP

During our pilot we received very positive feedback. In our initial phase we had created a MS Windows help-file that could be activated by pressing the Blaise help function when the cursor was on a specific question. The pilot government officials said that the procedure worked fine but it would mean an extra step on the part of the person entering data. They told us that the left column with the question showed a lot of unused space. Their suggestion was to use this area for help information. It looked easy and with the help of the Central Bureau of Statistics Netherlands we worked it out. Our MS Windows help file was quit extensive in information. Now all the information had to be put in the *.bla with its limitations.

We ended up using a lot of Aux-fields. We mean a lot because some help-text was just too long (more than 255 positions on one line). This is what it looks like in the rules after they are defined in the Aux-fields:

```
Ko08:=" is a keyword for the question, used in the printing function
Vr08:=" is the question
Df08:=" is the question definition
Tl081:=" is the question help text part 1
Tl082:=" is the question help text part 2
Tl083:=" is the question help text part 3
```

In the fields definition all these Aux-fields would come together in the question and it would look like this:

```
v08  "@B8@B ^vr08@/@/@G@BDefinitie:@B ^df08 ...
... @/@/@BToelichting:@B ^tl08^tl081^tl082^tl083@B@G" / "8" :0..365
```

Because we use PARALLEL in our *.bla we have a nice chapter overview through blocks. By using two include files, one of all the types and procedures together and one of all the tables we made a workable programming area.

The only thing that made the *.bla more complex looking then it really is: is the print function. With the help of Bas Weerman from CentERdata, he made a print_notepad.dll This file activates the MS Windows Notepad program. In this program, all the answers from one chapter are written; the question number, question keyword and the answers . This is the source:

The*.bla file in RULES section the following syntax is written:

```
{print.dll}
  Slot1.Keep IF Slot1 = Ja THEN Delete_File('open') ENDIF
  v03 IF Slot1 = Ja THEN Nette_Str(v03, temp)      Write_Ln(' 3 Dlt.avr. woningaanpassing-
voorlopige beschikking  : ' + temp) ENDIF
  NEWPAGE
  slot1
  slot11
  IF Slot11 = Ja THEN Slot1 := Nee ENDIF
  IF Slot1 = Ja THEN Open_NotePad('open') ENDIF
```

The Procedures.inc file in the procedures section the following syntax is written:

```
PROCEDURE Delete_File
PARAMETERS
  IMPORT
  Tekst: String
  ALIEN('print_notepad.dll',1)
ENDPROCEDURE
```

```
PROCEDURE Write_Ln
PARAMETERS
  IMPORT
  Tekst: String
```

```
    ALIEN('print_notepad.dll',2)
ENDPROCEDURE
```

```
PROCEDURE Open_NotePad
PARAMETERS
    IMPORT
        Tekst: String
    ALIEN('print_notepad.dll',3)
ENDPROCEDURE
```

```
{Definition type for the print_notepad.dll }
PROCEDURE Nette_Str
PARAMETERS
    aIndex: Integer
EXPORT
    aString: String;
RULES
    aString := "
    IF aIndex = EMPTY THEN aString := " ENDIF
    IF aIndex = DONTKNOW THEN aString := '?' ENDIF
    IF aIndex <> EMPTY AND aIndex <> DONTKNOW THEN aString := Str(aIndex) ENDIF
ENDPROCEDURE
```

The main reason for this print function was for the Government official to check if he made typing errors in a specific chapter. A print could also be given to a certain department that would provide all the data for a specific chapter to the Government official. He can therefore make a print of every chapter [Block].

At the end of each chapter there were two questions. The first question would activate notepad and place all the answers in a text-file. This could be printed at that moment. After that action the second question had to be answered to deactivate notepad and go to the following chapter.

This is a functionality we have to improve for next year because some government officials closed the DEP program after printing. This meant that the next time they opened the DEP program the first print question was still in the 'yes I want to print setting', and notepad was started automatically. I saw returned answers in which one Government official started printing after he had entered all his data. When I opened his DEP version 5 notepads were opened automatically and I had 5 prints of answers. This one item seemed confusing to some people. They probably do not read the print functionality correctly. Next year we will look at the implementation of alien routers.

4. The logistics of the benchmark

In the year 2000, 50 municipalities entered the Benchmark, in 2001 there were a 100 municipalities. For the year 2002 we expect a total of 150 out of 500 Dutch Municipalities who will participate in this great project. To make this benchmark efficient it needs good logistics functionality.

We adapted the experience and knowledge from Bas Weerman from CentERdata who operates a household panel in the Netherlands. Through the Blaise Api Calls a separate Blaise *.dbd is created for each Municipality through its unique ID-number. Then all the files are added and zipped to one file with this unique ID-number. Once these are emailed through our panel-emailing list we note email errors. A

confirmation from each municipality is required to fulfill the emailing list with a working DEP checkmark.

Returning emails from the municipalities is the reverse process. A program using Manipula adds all the separate *.dbd to one large file. Manipula is used again but this time to export the data to TAB ASCCI and read by a Cameleon SPSS import file into SPSS. A special report-module makes tables and graphs in MS Word document which will be emailed to all the participated municipalities.

5 Internet and the benchmark

In the beginning of the Blaise DEP programming, plans were made for putting the Municipalities Blaise data on the Internet at CentERdata. With this extra Internet functionality Municipalities can change their own data through the Internet by using their own special ID and Password. This functionality was being introduced when I was writing this paper. The benchmark has its own website for the Municipalities where they can change their address data and change their Benchmark data or add new data. During the presentation, I can tell more about the results of this part of the benchmark.

On the internet we also chose for the use of Parallel tab sheets and added an extra functionality: going to a the question right away. In a special section, the Government official can choose from a chapter pull-down menu the question-number with the question keyword. After the right selection, one is routed automatically to the specific question and the Government official can change his old answer or enter a new value. All of the Municipalities have a hardcopy of their answers from the DEP program. The question numbers resemble the Parallel tab sheets / chapter-numbers. Therefore changing or adding data through the internet will be relatively fast if you know which questions have to be changed or added.

This will give a similar look and feel as the Blaise DEP with the exception that on the Internet version one question at a time comes to the screen, except for the tables that appear as a whole. Next year we will ask a select group of Municipalities to use the Internet instead of the DEP program and ask for feedback. We found out that a lot of Municipalities use the printed version of the question to collect data from different departments. We would like to know next year if entering data through the DEP program is just as fast as using the internet.

At the special Benchmark website we added the reporting page. Each clustered group of Municipalities will arrive at there own cluster after using there login id and password. The report that is made through SPSS and shows it analysis in MS Word is archived in PDF format. Each cluster will see her own special *.PDF report and can therefore be easily downloaded if necessary. In the beginning the idea was to make a clustered report on the webserver but this would be too much time-consuming through all the different calculations that have to be made.

7 Future

The StimulansZ/SGBO Benchmark has been well received in Europe. The European Commission and the European Parlemtent have placed employment and social welfare programs as their international political goal. The Netherlands, Germany and Ireland with their specific programs have a yielding position in Europe on this topic. The CEMR organization is starting a benchmark-pilot for some major European Cities. A new development are the talks with the Dutch Central Bureau of Statistics CBS and StimulansZ for cooperation in the Benchmark.

8 Conclusion

The StimulansZ/SGBO Benchmarks will serve as outcome measures for the Municipalities results-driven approach to problem solving. Blaise is used for the Benchmark at its maximum potential: easy data-entry for non Blaise-users, namely Municipal-government-officials. They received through email a DEP program which was zipped. After unzipping the DEP program the Government Official could place the Benchmark program on a computer or network and the Government official could enter all the data easily through 5 chapters. Printing the results or emailing them back to the organization or changing the data through the internet were all that he could do to make the best of his data. Once the results were all printed in a report and emailed back to the Government official, or he could download it from the internet, the comparison of data started in the clustered Municipalities groups. Discussions among all the Government officials of different municipalities were started and its citizens will benefit from these results. Blaise is the main engine behind Data collection of the Benchmark and proved to be a major success.

**Blaise on Pentops, Blaise MetaData and Other Blaise Experience
in the National Health and Nutrition Examination Survey**

**David Hill and Christina Kass, Westat
and
Debra Reed-Gillette and Lewis Berman, CDC/NCHS**

Paper Not Available

Paper Not Presented at Conference

- Blaise to OLE-DB-Relational Data Migration? Yes!
T. R. Jellema, NAMES B.V.

Blaise to OLE-DB-Relational Data Migration?

Yes!

T.R. Jellema. Senior Consultant
NAMES B.V.
Guido Gezellestraat2
2394TT Hazerswoude Rijndijk
The Netherlands
TRJELLEMA@NAMESBV.NL

Yes!!!

This paper has its roots in an innocent question posed during the International Blaise User Conference 2000 to the Blaise Development team just after having been told about the exciting new capabilities of Blaise with regards to the capability to gain access to OLE-DB data sources. The question was deceptively simple: Were we (Blaise Users) going to see the OLE-DB equivalent of the ASCII-Relational export. The question had been anticipated well ahead. The answer in short was 'NO!!!'.

This article deals with the question and as suggested above provides an answer in the affirmative. Generating a generic database structure off a Blaise data model can be done programmatically and is quite straightforward after a spending a reasonable amount of time in getting to know the Blaise API. The utility program that we developed performs two tasks: 1) to generate a database structure, and 2) to use the Blaise API (rather than BOI files) to move the data across from the data model into the OLE-DB database.

Why bother?

Blaise is good, very good, in data capture and editing. Many statistical offices are quite happy to use Blaise and Manipula to process the data until the point where the data will be analyzed and tabulated using a statistical package. However more and more statistical offices have invested in the development of relational databases, and there is a need for transferring the Blaise data into these relational databases.

There is a perfectly workable solution for this problem. It requires you to first define the database structure that will receive the survey data. Then you will export the data from the Blaise database into ASCII tables that correspond with the structure of the relational database. Then batch uploading utilities such as Data Transfer Services (DTS) from Microsoft can be used to populate the relational database tables. The unfortunate part is that in case of complicated surveys, quite a lot of custom Manipula code needs to be written to generate the required ASCII tables.

The problem with this approach is of course that you will need to invest the time to define the database structure and subsequently you will need to write out the various Manipula scripts to write out the ASCII datafiles. It is time consuming to say the least. Also one might argue why it is necessary to use the ASCII format to generate an intermediate file – why not import directly from the Blaise database and -most of all- why not generate a database structure off the Blaise metadata? Certainly for those of us who have a passing knowledge of relational desktop databases such as Access, and who want a quick solution to our data-migration needs the solution presented above is cumbersome.

With Blaise III the problem had already been solved to some degree. It was (and is) possible to prepare a Blaise to ASCII relational export script using the Manipula setup wizard, and subsequently to use Cameleon scripts to generate setup files for the Oracle client server environment, or the Paradox (DOS version) desktop database. These setups could be used to generate the database tables and upload the ASCII relational data, and provided a workable solution. Unfortunately no Cameleon setups have since then been added to Blaise to support Windows databases. (although this is full well feasible). Therefore this solution was lost except to those few who were capable to develop their own Cameleon setups to do exactly what they wanted to achieve (such as generating a SQL script to define the metadata for a client server database).

The question that we pose in this article is whether the new capabilities on offer with the Blaise Component Pack (BCP 1.0) and Blaise 4.5 can be used to provide a significantly improved data migration tool.

Outline of a solution

What we would like to achieve is to be able to export a Blaise database to a relational database with the minimum amount of effort. This means that we will stick, for the moment, to the database structure implied by the ASCII relational data format.

The first task at hand is to programmatically define the database structure on the target database. Noting that we will present a generic solution for relational databases that are supported by OLE-DB (MS-Access, MS-SQL Server and Oracle), we would like to use the OLE-DB API to perform this task. The actual library used for this purpose is the ADOX library (Microsoft ADO Extension for DDL and security).

On the other side we will need to use the Blaise API to obtain all of the relevant information from the Blaise data model that we want to migrate to the database. We will need to make a correct translation for all of the various data types used in Blaise.

After defining the database structure, the next step that we would like to achieve is the actual migration of the data. Remember, we are interested in a one-stop solution. Here there are a number of choices. It is possible to migrate the data using the bulk import of ASCII relational data prepared by Manipula and the relational database utility for data import. We can also explore the use of BOI files (the Blaise to OLE-DB interface files) to import the data. Finally we can just use the Blaise API straight on to read Blaise data and to write out the relational tables.

In this paper we have chosen to explore the use of the Blaise API as a means not only to generate the database structures, but also as a means to populate the database tables. This is the closest we can get to a one-stop solution.

ASCII Relational Format

The ASCII relational format is a convenient and straightforward way of ‘unpacking’ a Blaise data model in a series of relational data tables. It was introduced in Blaise-III and is a logical development from the way storage was organized in older versions of Blaise. The basic organizing principle of the ASCII relational format is that each block type is stored in a separate file. Because there are potentially very many blocks in Blaise data models, the metadata definition supports the definition of *embedded* block types. These cause block type questions that have been embedded to be exported to the file representing

the enclosing block definition. If all block types are defined as embedded this implies that all the data is exported into a single file (ASCII export).

All questions that are defined as a blocktype are written out as separate records in the corresponding ASCII table. Each record written out corresponds to a particular *instance* of the block type. In each of the *enclosing* blocks the fields that correspond to a blocktype are written out as fields of type integer, and the *instance* reference is the value stored in these fields.

All of the ASCII tables produced by ASCII relational output have a unique identifier. The unique identifier is the primary key of the questionnaire, or a simple serial number if no primary key exists, and what is called an instance number, referring to the instance of the block type. If a data model has four fields that are defined as a block type, then the instance number refers to each of these fields. The instance number applies to all instances in the data model, and does not count local instances. In this way the unique identification of records in all data files produced by ASCII relational output is maintained in two fields.

The real advantage of the ASCII relational format is its closeness to the Blaise datamodel and the simplicity of the primary key data in the resulting tables. It is however a difficult format in which to define foreign keys and relationships between tables. Also it is often not practical to generate a separate table for each individual block and it is recommended to use the *embedded block* syntax to eliminate superfluous tables from the output.

Automated Data Definition and Data Migration.

The Blaise component Pack 1.0 offers programmatic access to Blaise data and metadata. With the component pack you will be able to access this information with any language that supports the Microsoft COM (Common Object Model) specification. This is actually quite a large collection of tools. (VB-Script, Visual Basic, Visual Basic for Applications, Delphi, C++, C# etcetera.). It will turn out that you will benefit from a programming environment that allows you to create your own object hierarchies and that allow you to use recursion. For our own implementation we have used Delphi. There are two reasons for this. Using Delphi we have been able to develop our own BCP aware components that can be used and re-used in any project. Also because of its support for pointer variables, it allowed us to define intermediate data structures that greatly facilitate the handling of the BCP data structures.

Exploring the Blaise Component Pack 1.0

The Blaise Component Pack consists of a number of libraries and controls. The Blaise API is the major part of the BCP. It exposes a large variety of objects that allow you to read almost any property or attribute that exists in metadata as well providing you with read and write access to Blaise databases.

For our purposes we need only tap a small amount of the capabilities of the Blaise API. We will be interested mainly in finding our way in the DatabaseManager object, the Database object, the Field object and the FieldDef object and the associated collections. When we were getting to use these objects we found the use of these objects not immediately intuitive. Therefore we will elaborate a little here.

Fields collections

The database object represents at the same time the datamodel (the Blaise metadata) and the database. In order to have a live database object you will have to use the Opendatabase method from the Database manager object.

From the database object we will need to access the various properties that allow us access to fields. There are in principle three distinct ways to do this

1. The Blaise datamodel has two properties (Fields and DefinedFields). By using either of these properties you can obtain a fields collection. (Basically a list of fields). The difference between the Fields property and the DefinedFields property is that the former contains all field *instances* in the datamodel and can be queried for data entered. The Definedfields property on the other hand yields a list of all defined fields in particular instances of block definitions and array questions and set questions that are *not instanced*. The fields and definedfields properties are indexed properties. This means that you will have to provide some index in order to obtain a Fields collection. For instance to obtain a collection of datafields you will need to provide a parameter blfkField:

```
FieldColl := bl4database.Fields[blfkData];
```

2. The Blaise datamodel has a Field property. By using the Field("Fieldname") property of the Datamodel object you can obtain a reference to the field object of that name. This presupposes that we have a list of field names at our disposal, and we know which is the field that we want to access.

```
aField := Bl4Database.Field['Ident.Person.Name']
```

3. Each Blaise Field has a Fields and DefinedFields property. With these properties you will get a collection of fields that are contained in a block, array or set question. You can use the Dictionaryasfield property of the datamodel to obtain a reference to a Blaise field representing the datamodel, or the topmost level in the fields hierarchy. You can use this field as a starting point for a *recursive* routine that will traverse all the hierarchical levels in the Blaise datamodel, and will expose all instanced and non-instanced fields in the datamodel using the Fields property that exists in each field object.

```
aField := Bl4Database.Dictionaryasfield
```

Fields and Blocks

However we are not just interested in obtaining a simple list of fields, or even a hierarchical structure of fields. First and foremost we need to have a complete list of all the block definitions in the datamodel. Because there is no Blocks property in the Blaise Database object, we will need to find each and every field that is a block definition. You can do this by using the Blaisefield.datatype property that should have the value represented by the blftBlock constant.

Unfortunately the Fields and DefinedFields properties from the Database object are not really useful for this exercise. The fields collection obtained from the fields property totally skips such fields, it only contains *instanced* fields. The fields collection obtained from the Definedfields property of the data model appears to be more promising, as all the defined blocks are neatly identified, and all of the fields definitions are clearly available. Unfortunately the Definedfields property does not provide a list of all possible instances of members of array fields. It is just too much work.

Using Recursion to get where we want.

The third option provided by the Blaise-API is an alternative and in our opinion better way in which to access the various block definitions and block instances. This is by using recursion on the elements of the fields property collection of individual fields. For instance let us investigate the case of an array field. The Blaise-API contains a Field object for the entire array question. In order to access the array members all you need to do is to obtain the fields collection of this field object, and deal with the individual fields in the collection. You can verify how this works by using the field selector component in the API that shows the various hierarchical levels.

Now whenever we encounter a block definition, we record it, and add to it each instance of a field with that block type. Each of the references is given an instance number. Subsequently for each of the (non-embedded) block definitions we trace out the fields of that block and any embedded clocks contained in it. For any field that is defined as a non-embedded block the type is changed into an integer field, it will contain the instance reference number of the block field instead, allowing for the relational structure. For each block definition we build an array of instances and each instance of the clock contains an array of the instanced fields contained within. Each instance of a block corresponds to a complete data record in a relational table except for the key information.

In order to be able to easily manipulate the information we are collecting recursively, we are actually generating a data structure that identifies all the block definitions, and for each block definition, (1) a list of all the fields that are defined in the block, and (2) a list of all of the instances of that block. Then for each block instance we also compile a list of references (pointers) to the actual Blaise fields. This is shown in Figure 1 below. When the time comes to process write out the database definition we will only need to iterate the Block Definition list (A), and for each of the block definitions, the block field definition list (B) to define the associated database table. For the actual data migration, each element in the Block Definition list (A) will be associated with a database table, and for each element of the block instance list (C) a record will be written out to the appropriate database table with the corresponding values contained in the block field instances array (D).

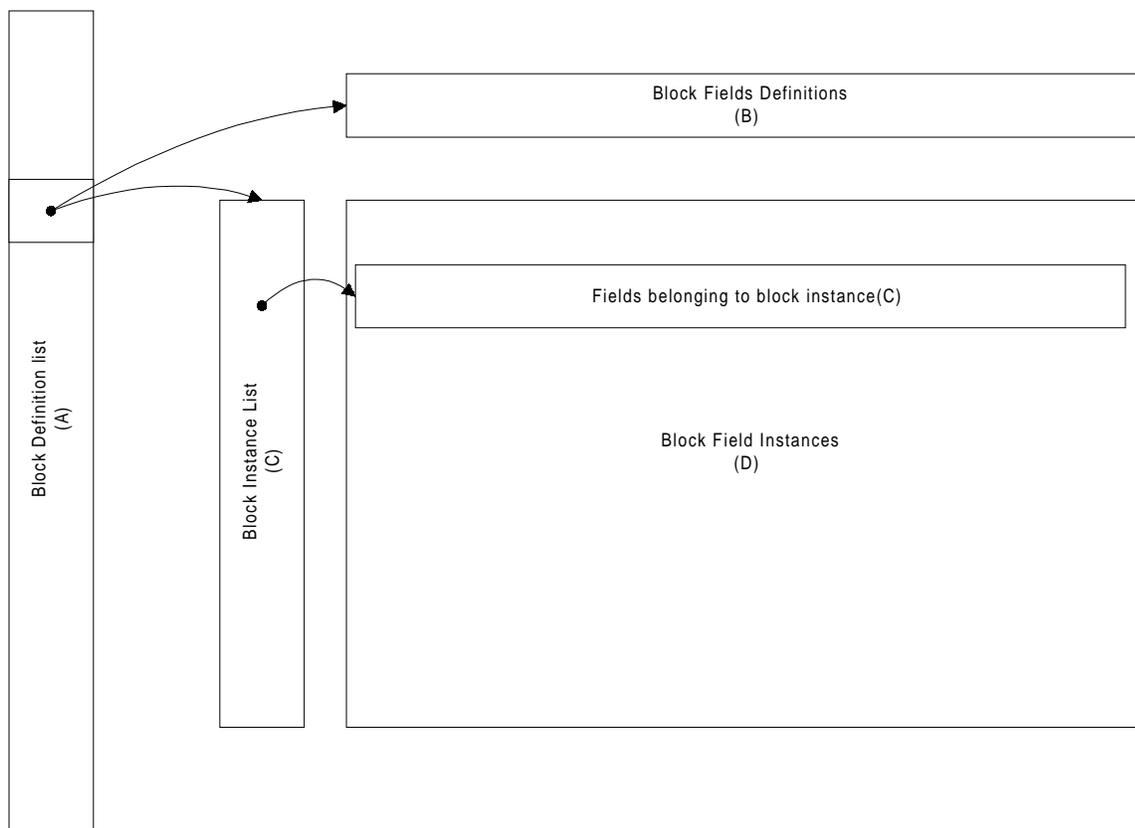


Figure 1 Intermediate Data Structure

The really nice thing about the ASCII relational format is that the key information is stored in two fields only. Each table has the fPrimary field, that is a collation of the primary key fields in the data model, or the internal form number of the questionnaire and each table has an instance number, allowing multiple instances of the same block occurring within the same questionnaire. This is true even for the data model table. When translating the data it is therefore not required to pay much attention to the construction of the primary key of detail tables. We can mimic this behavior in the Blaise API, as the database object provides a property that provides the text representation of the primary key value.

The drawback of the ASCII relational method is the existence of the instance fields in the data model record layout. For on, they will probably break certain database products that have an upper limit in the number of columns to a table. It is easy to define an array of 500 elements in Blaise. If these elements correspond to blocks, this means that the containing block or data model will have 500 instance fields. For a Database Administrator tables that contain large numbers of instance reference fields are quite ugly. It is quite hard to define relational integrity rules in such a structure.

Producing the Metadata.

Once we have obtained a list of all the different block definitions, and allowed for the existence of embedded blocks, it becomes possible to write out table definitions on the basis of the Blaise Metadata. The definition of the database structure in relational databases can be done by preparing a database script that contains the various statements containing the DDL (Data Definition Statements) for the relational database, or by using a special library, the ADOX library. (ADO Extensions for DDL and Security). The ADOX library provides a series of objects (catalog, table(s), field(s), index, key) that allow the definition

of all aspects of an OLE-DB database. Our task is merely applying the methods provided with these objects.

The most difficult part of this is finding the correct translation of Blaise datatypes into OLE-DB data types. Particularly difficult are date and time types. As you may know, the BOI file format does not support translation of Blaise data and time types into the OLE-DB date and time counterparts. Instead it translates dates and time into string information. This is because there are different OLE-DB data types for dates and different OLE-DB data types for time. For instance a DATE field in MS-Access has a OLE-DB data type `adDate` and a DATE field in Microsoft SQL server has a data type `adDBDate`.

The makers of Blaise have found the inconsistency in the treatment of date and time fields in OLE-DB so embarrassing that they actually do not support the translation of Blaise DATE and TIME types into anything else than text format.

On the other hand, it is quite cumbersome to first migrate data from Blaise containing dates that are transferred into a string format, and subsequently to have to convert these again in the appropriate types after migration of the data. It is perhaps more convenient to allow the user to choose which OLE DB Date and time type is appropriate. The other Blaise types are translated straightforwardly.

An intriguing option offered by the Blaise API is the inclusion of data in the OLE-DB tables that is actually only available at runtime, such as auxfields and parameters. This can be achieved by using the correct Blaise fieldkind selector set in defining the fields collections obtained from the bock instances. Optionally it is possible to write out such information as well.

Exporting the data.

After having defined the database structure in the OLE-DB database along the principles of the ASCII-relational output, it becomes necessary to actually export the data. Here again we have a choice as to the implementation.

We could use the ASCII file format as an intermediate file format, and use the bulk read options provided with the target database software or we could specify a BOI file for each table, and generate MANIPULA code for each of the tables. Alternatively we could attempt to write the information in the database out to the OLE-DB database directly using the Blaise-API functions. The latter option appears to be more elegant and is the one that we have implemented.

In our migration program we need to define an ADO recordset object for each block type definition that does not involve embedded blocks. These link into the various tables that were defined using ADOX. Because of the potentially large volume of changes that will be made to the data, it is decided to cache the data in memory (using ADO) and to write out the changes for every 100-200 Blaise data records.

Then the program iterates through all of the Blaise database records, and for each of the block types (that are not embedded) it writes a record to the corresponding database table for each instance.

The program: OLE-DB Relational Export.

Interface

The program has a quite simple user interface. It consists of a number of screens that can be accessed through a tabbed notebook. The first two screens are purely informational, these are the structure view and the defined fields view. The Block structures view actually shows the contents of the data structure that is diagrammatically presented in Figure 1. There is also an options page that allows you to set some preferences, such as choosing the correct Datefield data type.

In the structure view the program shows the hierarchical structure of the data model. This tree view shows virtually the same information as the Blaise field selector object included in the BCP 1.0. However because of problems with we experienced with the BlaiseFieldSelector we opted to develop our own component. The tree view shows the structure of an example data model with a large amount of block definitions.

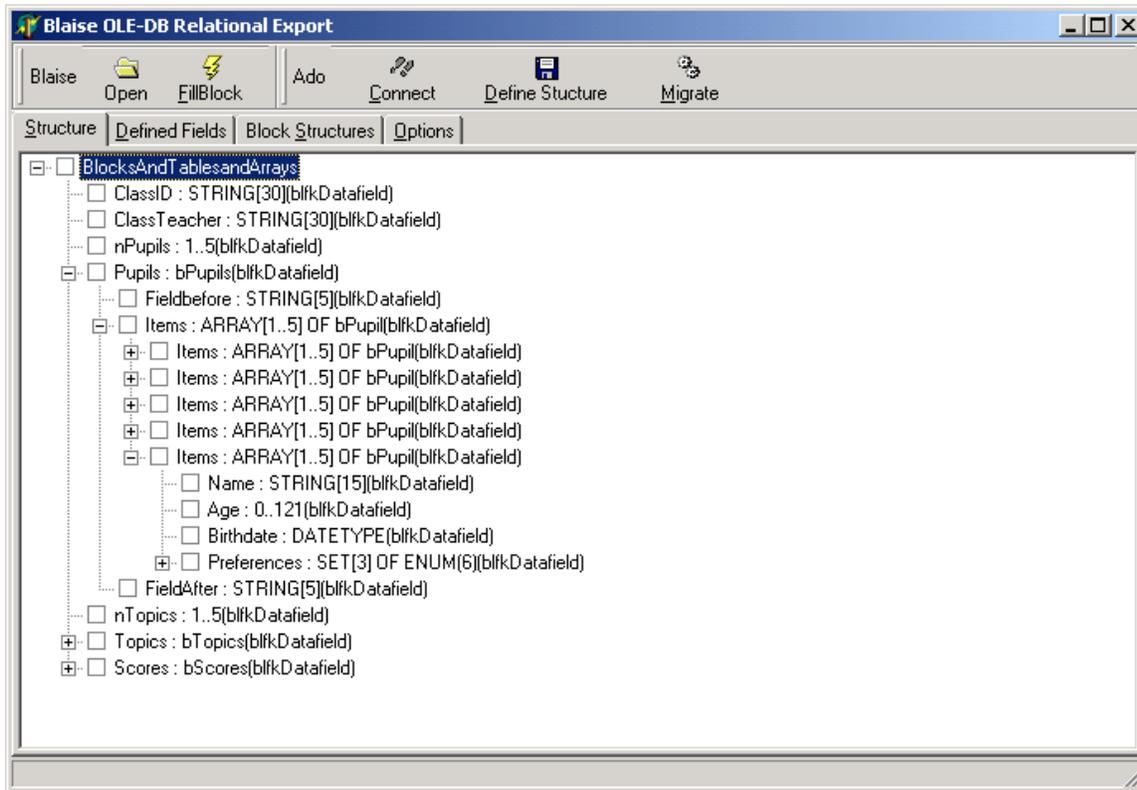


Figure 2 The Data model Structure View

In the Block Structures View you can see a tree representation of the various BLOCKS that have been defined, and the definition of the fields contained in each block, as well as the instance references for each block. For each of the blocks shown in the Block Structures view, a database table will be created. When migrating, for each instance encountered of such a block, a record is created in the database table.

Here you can see that the block bPupil contains information on name, age, birthdate and preferences. You can also see that the pupil block is instantiated five times in the array Items.

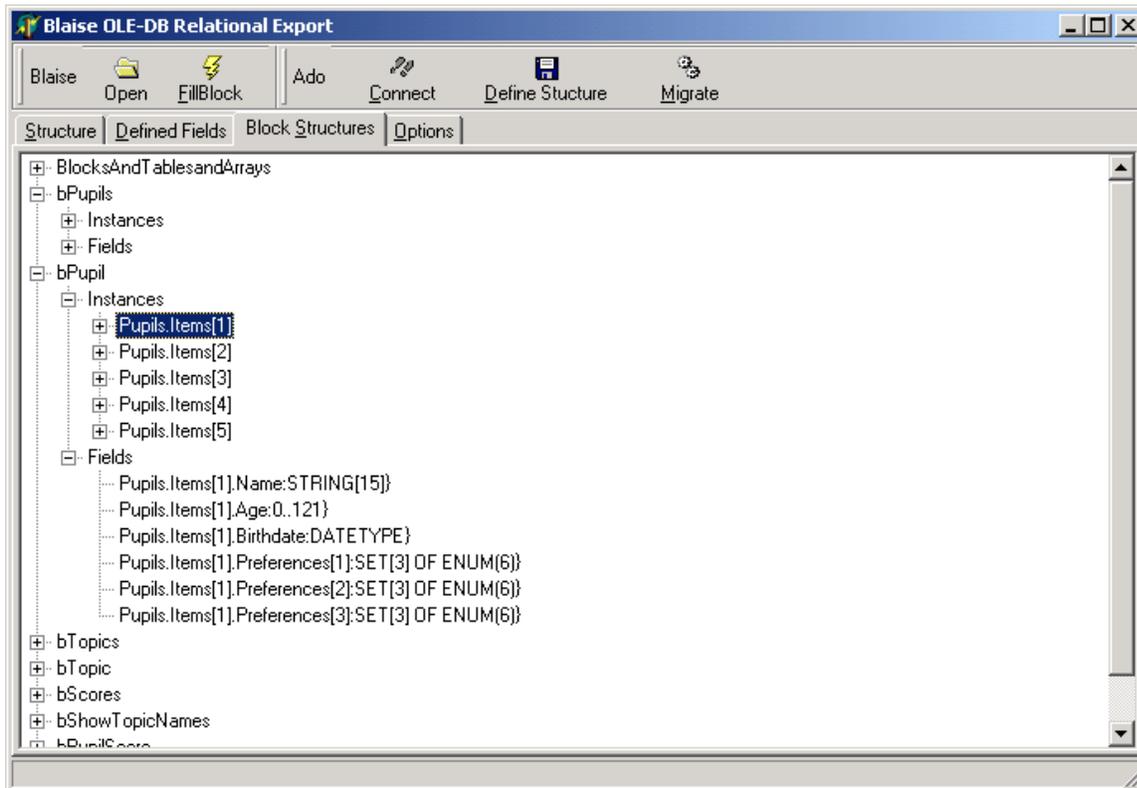


Figure 3 The Datamodel Block Structure View

The block structure view therefore allows you a complete overview of the relational structure that will be created.

To operate the program is quite simple, it is a matter of consecutively defining the following information:

1. Press the 'Open' button to open the Blaise database and data model.
2. Press the 'Fillblock' button to generate the block structures
3. Press the 'Connect' button to open an existing OLE-DB database
4. Press the 'Define structure' button to generate the database structure
5. Press the 'Migrate' button to transfer the data from the Blaise database into the relational database.

Data types and conversions

In the program most Blaise datatypes end up translated as you would expect it:

Table 1 Conversion of Datatypes

BLAISE Datatype	ADO (OLE-DB) datatype
INTEGER	AdInt
REAL	AdDouble
STRING	AdVarWChar
CLASSIFICATION	Advarwchar
ENUMERATION	AdInteger
DATETYPE	AdVarWChar AdDate AdDBDate
TIMETYPE	AdVarWChar AdDate AdDBTime
OPEN	Not supported
BLOCK	AdInteger

With regards to the translation of Date and Time fields you have a choice, you can specify this in the options page.

Performance

The program provides an easy and complete way of migrating data from *any* Blaise datamodel to *any* relational database that is supported by the OLE-DB standard. This gives a user a great amount of flexibility. The migration of the data is however prone to take a long time; the combined performance of the Blaise API and the ADO libraries is less than inspiring. It is probably faster to export data to ASCII and subsequently import it into the relational database than to perform the task directly using the API approach. This is due largely to the overhead incurred in adding data record by record into an OLE-DB table. As mentioned earlier, we found it quite necessary to cache large amounts of data in memory rather than adding data, record by record, to the relational tables.

BOI

One of the major new features in the BCP and Blaise 4.5 is the ability to pull in data from OLE-DB datasources into Blaise. Although it is possible, using a Manipula script, to pull data out of a datamodel into a relational table, it does not seem that BOI files are designed for this task. One indication of this is the absence of Blaise to OLE-DB and/or an ASCII to OLE-DB option in the Manipula Wizard. Another indication is that the Blaise OLE-DB Interface Wizard only allows you to define a BOI file based on existing database tables; but does offer you to generate a Blaise datamodel that conforms to the database table layout. Although it is possible to use a BOI file that has been set up to allow relational data to be brought into Blaise to pull data from Blaise into a relational table this is cumbersome in the context of relational data. You will need to specify a BOI file for each target database table. Subsequently you will need to write a Manipula script that will parcel out all of the required data into the component tables of the relational database. Not a one stop solution!

Note: Model-language block tagging.

In one of the presentations of the previous IBUC Pierzchwała has suggested using a language definition in the data model (language MDL) to indicate which blocks should be included in separate relational tables. In the metadata definitions, each (non-embedded) block that has an MDL description is written out to the table corresponding to the Block description. The method is flexible, because (as does the EMBEDDED keyword) adding such description items will not break the data model and these descriptions may be provided at a later date without requiring migration of data to a new data model.

Pierzchwała does however not indicate in his article how he deals with identifying instances or how the primary and foreign keys are defined on the basis of this information. It is however possible to define database structures on the basis of model language block tagging once rules are defined about the construction of relationships between the various database tables. The advantage of this method can be that block instances of the same type are assigned different model languages, and therefore will be included in different database tables. For instance address information can be collected on the respondent as well as the place of work of the respondent. It may be that such information needs to be separated into two tables. Another case would occur if instances of different blocks would be assigned to the same MDL tag. This would mean that the table structure corresponding to the MDL tag would contain the fields from both blocks.

With a satisfactory solution to the definition of primary keys and foreign keys for the relational tables the model language block tagging technique can be implemented in an equally straightforward way as the ASCII relational format presented in this paper.

References

1. Blaise 4 Windows, Developers Guide
2. Blaise Component Pack 1.00, Beta Version, Helpfile
3. Programming Access 2000, Rick Dobson, Microsoft Press.

