

11th International Blaise Users Conference



*Conference Proceedings
Annapolis Maryland
September 2007*

**Proceedings of the 11th
International Blaise Users Conference**

IBUC 2007

**Annapolis, Maryland USA
September 26 – 28, 2007**

Preface

This document contains the papers and posters presented at the 11th International Blaise Users Conference held at the Doubletree Hotel Annapolis in Annapolis, Maryland, USA, from September 26th to 28th, 2007. The conference included three days of technical papers and presentations on the use of Blaise and related topics.

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

- Vesa Kuusela (Statistics Finland, Chair)
- Bill Connett (The University of Michigan)
- Gina-Qian Cheung (The University of Michigan)
- Karen Bagwell (US Bureau of the Census)
- Rebecca Lui (Statistics Canada)
- Mark Pierzchala (Mathematica Policy Research, Inc., USA)
- Jim O'Reilly (Westat, Inc.)
- Lon Hofman (Statistics Netherlands)
- Tim Burrell (ONS, UK)
- Pavel Kozjek (Statistics Slovenia)

The University of Michigan's Survey Research Center, as the host organization, has collected, edited, and printed these proceedings for the benefit of the conference participants and others.

IBUC 2007 was organized and hosted by The University of Michigan's Survey Research Center. The organizing committee consisted of:

- Gina-Qian Cheung
- Jim Hagerman
- Ashanti Harris

Table of Contents

Using Blaise 4.8

Encrypting Blaise Data on Network Servers	1
Generic Data Storage with Blaise 4.8 Datalink.....	7
Methods for Simultaneous Meta & Data Manipulation in Blaise	31

Blaise & Methods of Documentation

Blaise and Xml: Experiences Of An Outsourcing Project.....	39
Michigan Questionnaire Documentation System (MQDS): A User's Perspective	51

Blaise Programming

The use of mobile devices to carry out Blaise surveys at the Office for National Statistics	67
Using the Blaise Alien Router for Audio-Recording of In-Person Interviews	77
Dynamic List Building Using Blaise	89

Testing Blaise Surveys

CTT: CAI Testing Tool.....	99
Blaise Testing.....	109
Case Scenario Library for Testing Large Blaise Applications	117

CATI & CATI Management

CATI MANAGEMENT: Behind the Scenes.....	133
Computer-Assisted Dialing: What will it do for you?	145

Audit Trails

Using audit trails to monitor respondent behaviour in an Audio-CASI questionnaire	155
---	-----

BCP

Experiences Using an Event History Calendar in the Panel Study of Income Dynamics	165
Retrospective Data Collection	179
Navigating BCP with .NET	191
Lifecycle Processes to Insure Quality of Blaise Interview Data	217
The ScriptWriter Tool - An Application for Interviewer Training Script Development	231

Blaise Questionnaire Specification, Design, and Implementation

Challenges in Converting the National Crime Victimization Survey to Blaise	241
The ONS Integrated Household Survey: The next installment	257
Questionnaire Specification Database for Blaise Surveys.....	269
Authoring Blaise questionnaires – a task for the survey specialist or an IT programmer?.....	279
Blaise Source Code Editing System.....	293
BLAISE in Macedonia	349
Designing and Developing Person-based and Topic-Based Data Collection Instruments.....	357
Coping with people who just won't stay put: The Use of Blaise in Longitudinal Panel Surveys	367
Exploration of Blaise Instrument Generation from Metadata	385

Using Blaise 4.7 for data entry, checking, error reporting and transforming for further processing.....	395
Survey Specifications Management at Statistics Canada.....	401

Blaise and the Internet

Analyses of Web Survey Data	407
Web Application Stress Testing with Blaise IS	413

Poster Sessions

Conversion of Blaise Databases to Relational Databases*	423
Deploying a C# .NET Object onto a Laptop to be used with a Blaise 4.7 Data Collection Instrument.....	439
Designing an e-Form to Collect Survey Data	447
Miscellaneous Examples of Programming in Blaise and Manipula.....	459
New CAI-operation at Statistics Norway.....	469

Contacts..... 477

Encrypting Blaise Data on Network Servers

John Mamer, Leonard Hart and Josh Rozen, Mathematica Policy Research, Inc.

1. Background

Identity theft and privacy issues have brought increased scrutiny of the technology and procedures by which confidential data about individuals are protected. High-profile cases over the last several years, in which confidential data have been stolen or inadvertently released, have dramatically increased both awareness of these issues and pressure to take all conceivable precautions to prevent such incidents. These cases and the increased scrutiny they have brought to security and confidentiality issues have affected virtually every public and private institution in the United States.

One of these cases—the theft of a laptop containing personal records of more than 26.5 million veterans from the U.S. Department of Veterans Affairs (VA)—brought increased focus by agencies within the federal government on the issue of data “at rest.” The level of protection implied by this concern goes beyond the two issues that have plagued the information technology industry since the inception of the internet: (1) security of data transmitted across the internet (“in motion”), and, (2) unauthorized intrusion into local area networks maintaining secure data (“hacking”). Security directives from a number of government agencies—backed by National Institute of Standards and Technology (NIST) guidelines—are increasingly calling for the encryption of all data stored on electronic media.

Initially, federal government agencies focused their attention on the vulnerability of laptop data exposed by incidents such as the VA theft cited above. However, they are now expanding their focus to network servers and other network-connected devices behind secure network firewalls. Mathematica Policy Research, Inc. (MPR) collects data—often very sensitive data—on behalf of the federal government and security has always been a major concern. New security directives from the federal government impact the systems we develop and/or use to collect such data. Since Blaise is one of those systems, we have investigated the means by which Blaise data can be held in encrypted form. This paper presents the results of our investigation.

2. Full Disk vs. Content Encryption

Our investigation into encrypting Blaise data began with identifying the differences between full disk encryption and content encryption. An understanding of these differences is important to devising an approach to Blaise encryption. In this section, we summarize our findings.

2.1 Full Disk Encryption

In full disk encryption, the entire contents of a data storage device are kept in an encrypted format. Full disk encryption is most easily understood in the context of the theft of the VA laptop noted above. Even with strong passwords in use at the operating system level, data on an unencrypted drive might easily be accessed by simply mounting that drive as a non-system drive on a different computer.

A fully encrypted hard drive renders the laptop's drive useless to any user who does not have the username and password (or encryption key). Authentication components of the software are installed in the system drive's Master Boot Record (MBR); thus there is no way of circumventing the authentication process when the laptop is booted. When a valid username and password are supplied, the drive contents (starting with operating system files) are decrypted into memory as needed. Data are written (encrypted) and read (decrypted) to and from the drive via what amounts to system level i/o drivers, transparent to users and programs. Other than a small performance penalty, the machine runs as it normally would.

It is now standard practice at MPR to run full disk encryption (256-bit AES) on all corporate laptops. As a priority, we encrypted all laptops used in our Computer Assisted Personal Interviewing (CAPI) applications. The CAPI laptops run the Blaise software along with SurveyTrak software developed by the Survey Research Center at The University of Michigan. Identifying information on the respondents to be surveyed, as well as information gathered from these respondents using the Blaise system, is stored in a Sybase database in encrypted format. This provides reasonable protection to data on the laptops. However, the Sybase encryption—at least the version of Sybase in the SurveyTrak implementation used by MPR—does not conform to the Federal Information Processing Standards (FIPS) 140-2 certification requirements from the NIST. Adherence to FIPS 140-2 (Security Requirements for Cryptographic Modules) is now being required by many of our clients.

MPR evaluated two products for implementing full disk encryption on its CAPI laptops—SafeBoot (from SafeBoot International) and Pointsec (from Pointsec Mobile Technologies, Inc.). Both worked as advertised and would have met all government standards with one exception—Pointsec was not FIPS 140-2 certified for the version of Windows that was being run on our older CAPI laptops. Primarily for this reason, we eventually chose SafeBoot.

Neither Pointsec nor SafeBoot was designed with network server drives in mind, but follow-up discussions with sales people from the respective organizations indicated that the products would work in the network environment. However, for the purposes of these applications, the Network Operating System (NOS) is the “user” of the network drives. Once the NOS supplies the password, just as with laptop drives, the encryption of data becomes transparent to the “user.” While this is certainly good from the standpoint of ease of implementation, in the network environment the only event that full disk encryption protects against is the theft of a network server drive. This limitation led to the investigation of content encryption, as described in the next section.

2.2 Content Encryption

The term “content encryption” appears to be trademarked by SafeBoot but is often used in conjunction with the products supplied by a number of other vendors, such as PGP, McAfee, and Coreguard. These products vary in their details but share a few common features:

- Data can be encrypted on network drives at the file or folder level

- Authentication through an “encryption engine” or an “encryption server” is required before data can be passed to a network user
- Client-side software is used to decrypt the data when it is accessed on the user’s machine, again requiring authentication when decryption is performed

The disadvantage of content encryption is the significantly increased cost of purchase, implementation, and administration compared to full disk encryption. The advantage is that content encryption provides protection against much more than the theft of a network server drive, including the following types of protection:

- Data are encrypted as they are transmitted across the local area networks
- Data can be protected even from network administrators
- As one vendor (PGP) puts it, “the encryption follows the data.” For example, data can be downloaded from a network drive to a USB drive by an authenticated user but it will be in encrypted form. It can subsequently only be accessed by a user who is authenticated through the encryption server.

3. Blaise and Network Server Encryption Technologies

With the previous section as background, this section describes our investigation into the specific encryption technologies that we considered for use in storing and processing encrypted Blaise files on network servers.

3.1 Windows Encrypted File System

The Windows Encrypted File System (EFS) from Microsoft is best classed as a content encryption approach to encryption. As with many “free” Microsoft products, it does not provide much of the functionality that third party vendors provide but it has some of the attributes described above for content encryption. EFS can be used on a desktop with the Windows operating system (XP or higher is best) or on Windows-based servers to encrypt individual folders or files. After encryption, only authorized users may access or decrypt the files.

Our initial expectation was that EFS would provide the best approach to encryption for Blaise files. This was based on our success using EFS for encrypting SQL Server databases, which our federal government clients are also requiring. There are three salient aspects of utilizing EFS for encrypting SQL Server databases:

- An SQL Server database is a single file in the Windows operating system and EFS is easily applied to that single file.
- The SQL Server database engine is the “user” of that file. Once SQL Server has performed the decryption, all programs run transparently against the database (that is, no code changes are required to switch from running an unencrypted database to running an encrypted database).

- Performance penalties are documented in the 5 to 10 percent range and we observed no appreciable degradation in response times for online, interactive applications using an encrypted SQL Server database.

For encrypting Blaise applications, it quickly became apparent that EFS was *not* a workable solution. Two problems presented themselves immediately:

- With EFS, users can encrypt data at the file or folder level, but they can only grant access at the file level. Whereas with SQL Server we only had to encrypt a single file, with Blaise many files are involved: the .bdb file, audit-trails, batch and overnight files, lock files, temp files, and so on. We determined that it was not feasible to administer individual file rights under EFS to multiple Blaise files when we have many Blaise applications being implemented in a constantly revolving environment.
- With SQL Server—a true client server application—only one “user” needs to be authenticated in order to provide decrypted data to multiple application users. That user is the SQL Server application itself. With Blaise, each individual Blaise user (interviewer) requires authentication, again adding to administrative burden.

3.2 Novell Network Operating System Version 6.5

MPR has utilized Novell as the platform for its Blaise applications. Novell does not have the security vulnerabilities associated with Microsoft, largely because it is not as ready a target for hackers. Also, Blaise utilizes a network server as a file server and Novell is a very efficient file server. Market pressures and other factors will, in the long run, force MPR to move to the Windows server platform but this will be a difficult migration. Even if we had found Windows Encrypted File System to be a good solution to the encryption problem with Blaise, the difficulty of migration and the need for a quick solution to the encryption requirements would have presented a major challenge.

Version 6.5 of the Novell Network Operation System (NOS) provides for the encryption of network drives. Our initial tests show that performance of Blaise on encrypted network drives utilizing Version 6.5 of the Novell NOS is acceptable. Our plan is to migrate any Blaise application requiring encryption on network servers to a server running Novell NOS Version 6.5. These applications and surveys for the most part are currently running in Blaise Version 4.7. In this particular approach, Blaise Version 4.8 would appear to provide no particular advantage, so we are not planning to utilize Version 4.8 as a part of the solution to Blaise encryption.

3.3 Content Encryption

In the long term (but it is not too far away), we expect that the federal government will require content encryption for all confidential data including any data in Blaise files or related processes. All such files will need to be encrypted using some form of content encryption. This will secure data that are transmitted across the local area network, it will protect data from network administrators, and it will protect data that are downloaded onto removable media. Unlike Windows EFS, which only allows users to grant and revoke rights at the file level, commercial content encryption packages allow these facilities at the folder level. With these facilities, the administrative burden will be significant but it should be manageable.

3.4 Encryption and Decryption Using the Blaise Software

The feasibility of storing Blaise files as encrypted files, and handling encryption and decryption within the Blaise software, was raised with the Blaise team at the Blaise Corporate User Group meeting held in Dusseldorf earlier this year. The team's response was that this was feasible from a development standpoint but that it could seriously impact usability and ease of implementation of Blaise applications, because of the login and password requirements that would be required for both interviewers and application programs (written in Manipula or any other application) that access Blaise data. Based on this discussion, it was decided to drop this requirement from the "wish list" for future Blaise releases. Given the clearer definition of requirements that we feel we have gained over that last six months and the administrative costs of the content encryption approach that is likely to be required in the future, we feel that it would be worth reconsidering this option, but, given that content encryption may be "in our future" for many other reasons, encryption through the Blaise software may still be deemed a less attractive option.

4. Conclusions and Next Steps

As noted above, MPR believes that content encryption of all confidential data, including the data used to field a Blaise survey and the data collected by a Blaise instrument, will be required in most or all federal government applications/surveys. MPR will continue its investigation of third party vendors providing content encryption solutions. Once a vendor is selected, the implementation of encryption in Blaise will be a difficult conversion where not only the .bdb file but all files related to Blaise processing containing confidential data will have to be reviewed. By segregating the .bdb file to a separate server, Blaise Version 4.8 may help in the solution of the encryption problem and this will be investigated. We would welcome a review by the Blaise team of the requirements and issues presented in this paper in order to reconsider whether an encryption/decryption facility within the Blaise components presents a better option. However, in sum, the impact of new security requirements on the Blaise application and any other applications that store and process confidential data should not be underestimated.

Generic Data Storage with Blaise 4.8 Datalink

Arno Rouschen, Statistics Netherlands

1. Introduction

In Blaise 4.5 / BCP 1.0 it was possible to read / write data in external 'data stores', for example, relational databases and Excel spreadsheets. This is realized by using Microsoft's OLE DB technology. The concept of Blaise working together with OLE DB data sources is called the Blaise Datalink.

From Blaise 4.6 / BCP 2.0 on it became also possible to store Blaise questionnaire data in relational database management systems (RDBMS), like Oracle and Microsoft SQL Server. Within this and higher versions of Blaise you can use such a BOI file in the same way as a native Blaise database file. Blaise OLE DB wizards are available to create a BOI file and the corresponding database tables for any Blaise questionnaire. The resulting BOI file contains connection parameters and information about available tables and how to retrieve field data from these tables.

In Blaise 4.7 Enterprise Blaise Datalink has become a mature technology that can be used in both regular or internet survey scenarios. With Datalink it became easy to run an internet survey on a Blaise IS server and to store the collected data in a secure relational database server behind your firewall. BOI files can now be used to access the most popular databases, but also Blaise data files and even ASCII files.

Datalink has evolved further in Blaise 4.8. Although it is already a nice feature that you are able to store Blaise questionnaire data in an RDBMS, we had some ideas and also had some requests from clients to make the storage more generic; it would especially be nice if we could share database tables between questionnaires. In previous Blaise Datalink aware versions, BOI files were tailor-made for a particular survey and each Blaise questionnaire had its own set of database tables.

Blaise 4.8 Datalink has been extended and can define, next to the tailor-made BOI files, so called generic BOI files. Generic BOI files open the way to a centralized input data store; all survey data can be stored in a common set of tables in a relational database.

Another new feature of Blaise 4.8 Datalink is versioning. You can now have more versions of the same record in the database. This opens many opportunities, as you will see later in this paper.

2. Generic versus non-generic BOI files

If you create a BOI file for a Blaise dictionary then several table definitions will be created in order to store Blaise record information in a relational database. This information includes questionnaire data, form status information, answers to open questions, remarks and other data.

Until now these tables were strongly tied to the data model that was used with the BOI file; column sizes depended on the exact field sizes as defined in the data model, primary key fields defined in a Blaise data model were also used in the database tables and only required data columns were included in the database tables. The table names depended

on the name of the data model, i.e. BOI table definitions were non-generic and tailor-made.

These non-generic BOI files are still supported in Blaise 4.8 (in fact they are the default), but it is now also possible to define so-called generic BOI files. The primary goal of these generic BOI files is to share tables between data models as much as possible. Generic BOI files have generic table structures which can be used to store data of multiple data models.

2.1 Generic BOI files in Blaise 4.8

If you intend to store Blaise questionnaire data in a relational database and also want to store this data in a generic way, then it is still required to create a BOI file for each Blaise dictionary that you use.

Data partition types, which are introduced in Blaise 4.6 and BCP 2.0, can also be used with generic BOI files. A data partition type defines in which structure data must be stored. We have the following data partition types:

- Flat; no blocks each field in your data model has a corresponding column in a table.
- Flat; blocks same as above, but now there will be a separate table for each block definition.
- In depth data is stored in a field – status – value way. Each data type has its own column.
- In depth text same as in depth; except that field values are stored as a string
- Stream stores the data of a record as a stream per block

It is perfectly okay to create a generic BOI file which uses the in depth data partition type for data model A and another generic BOI file which uses a flat storage structure for data model B. The added value of using generic BOI files is that most tables can be shared between the data models

Here is an illustration how table access works in both non-generic and generic scenarios. In case of generic BOI files database tables will be shared if possible, while in the non-generic situation each BOI file has its own set of tables.

Non-generic versus generic

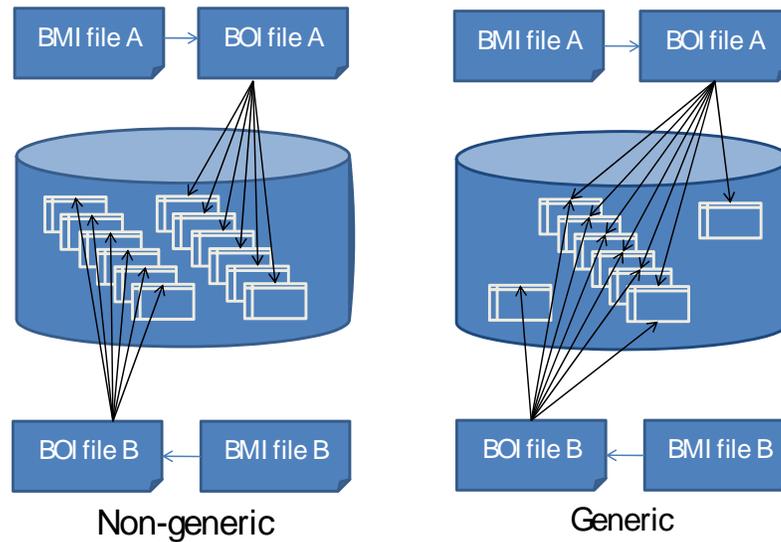


Figure 2.1 Table access with both non-generic and generic BOI files

2.1 Generic BOI files purposes

The main goal of generic BOI files is to share database tables as much as possible between Blaise data models. Generic BOI files can be used to setup a centralized data store. This is possible because these BOI files define tables which can be used with any Blaise data model and as a result have the possibility to store data from multiple surveys.

In such a scenario Blaise questionnaire data is stored in a centralized way in only a few predefined and fixed table structures. Because the number of needed tables decreases, the maintenance burden for your system and database administrators decreases also. The less tables the better is the motto.

Another advantage of using generic BOI files is that it becomes easier to embed Blaise cases and data within your own systems, because the generic BOI tables have a fixed predefined structure, which makes it possible that each survey can be linked in the same way. The concept of generic BOI files will be explained in more detail below.

2.2 Generic and non-generic tables

If we take a closer look at the tables which are created in generic BOI files, then we can also distinguish between non-generic and generic tables. Generic tables have always the same structure no matter which data model is being used. Because they have always the same structure, they are perfect candidates to be shareable.

Here is a listing of the generic tables, which are currently used by Blaise Datalink:

- Dictionary information table; stores the dictionaries which are part of the input data store
- Case information table; stores the cases of a survey
- Form information table; contains among others the form status information
- Block information table; contains the check status information of each block
- In Depth data tables; stores data in depth in a field – value way
- Remark table; contains the remarks with fields
- Open table. Table to store answers to open questions

There are just two types of tables who aren't generic. An example of such a table is a flat data table; in a flat table each column is mapped to a particular Blaise field of your data model. Needless to say that these tables only can be used with the data model on which they are based.

Example flat data table:

```
DATAMODEL test
```

```
FIELDS
```

```
    A: integer
```

```
    B: integer
```

```
ENDMODEL
```

If you create a flat data table for this data model then the data table will contain a column for field A and another column for field B.

The other non-generic table type is the Key information table. This is an auxiliary table which Blaise Datalink uses for retrieving records according to a Blaise key. Key information tables will be created in case of in depth and stream data partition types, because these partition types don't provide fast access to individual Blaise key field values. In these cases Blaise key fields are stored redundant; in depth or in a stream depending on your data partition type and also in a flat structure in the Key Information table.

I want to emphasize that despite the fact that flat data tables and key info tables cannot be shared between data models the other tables still can. If you open a generic BOI file in OLE DB Workshop you can see which tables are shareable.

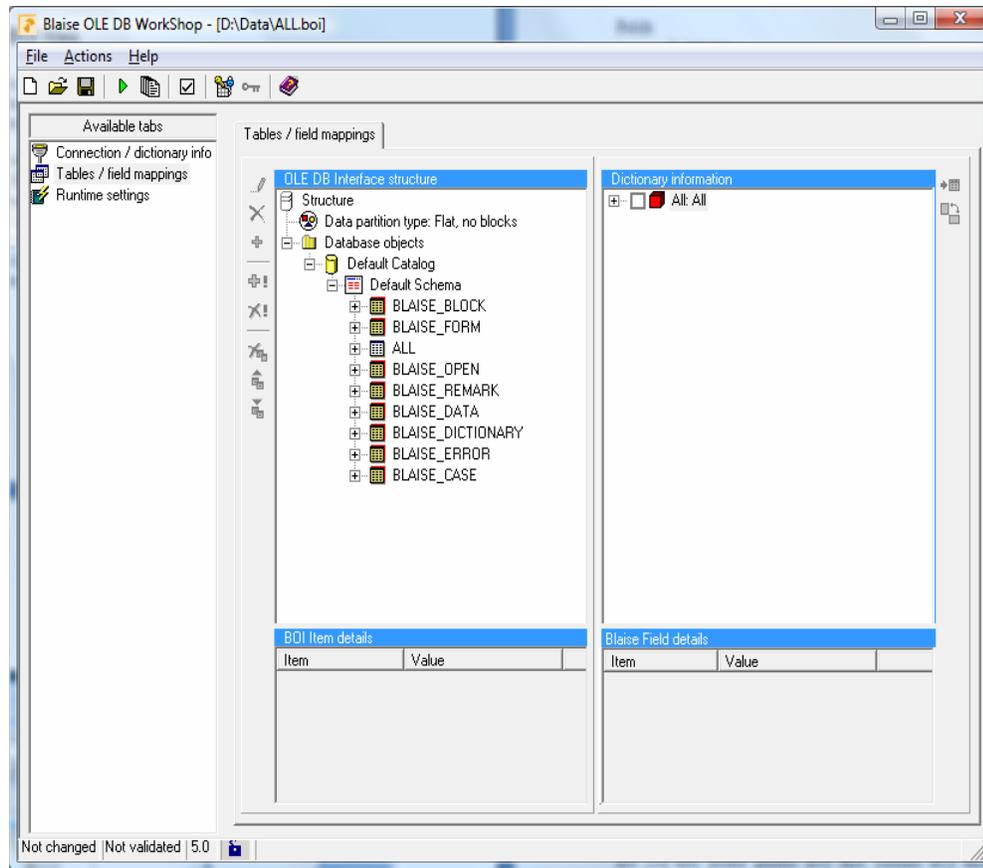


Figure 2.2 BOI Table definitions in OLE DB Workshop

Shareable tables have another (yellow) icon. By default their names begin with “BLAISE”. In this example there is only one non-generic table, table ALL. This is because this is a flat table with field mappings, i.e. each column in the table maps to a Blaise field of the associated data model.

See the appendix for more information about the available table types and their predefined structures.

3. Generic BOI files concepts

Generic BOI files use the following three important concepts, which allow us to store data from multiple data models in the same tables:

3.1 Common primary key

The most important difference between non-generic and generic BOI files is the primary key which they are using for their database tables. Non-generic BOI files use the primary key fields of the data model as primary key columns in the database tables. Generic BOI files on the other hand use a fixed and predefined primary key which is common to all generic BOI files.

This common primary key has the following columns:

JOINKEY	Contains a unique integer which identifies a case.
DMKEY	Contains the data model key
BEGINSTAMP	Contains the begin time of the period of validity of a particular record. This value has to be unique in combination with JOINKEY and DMKEY. This column is used for versioning.

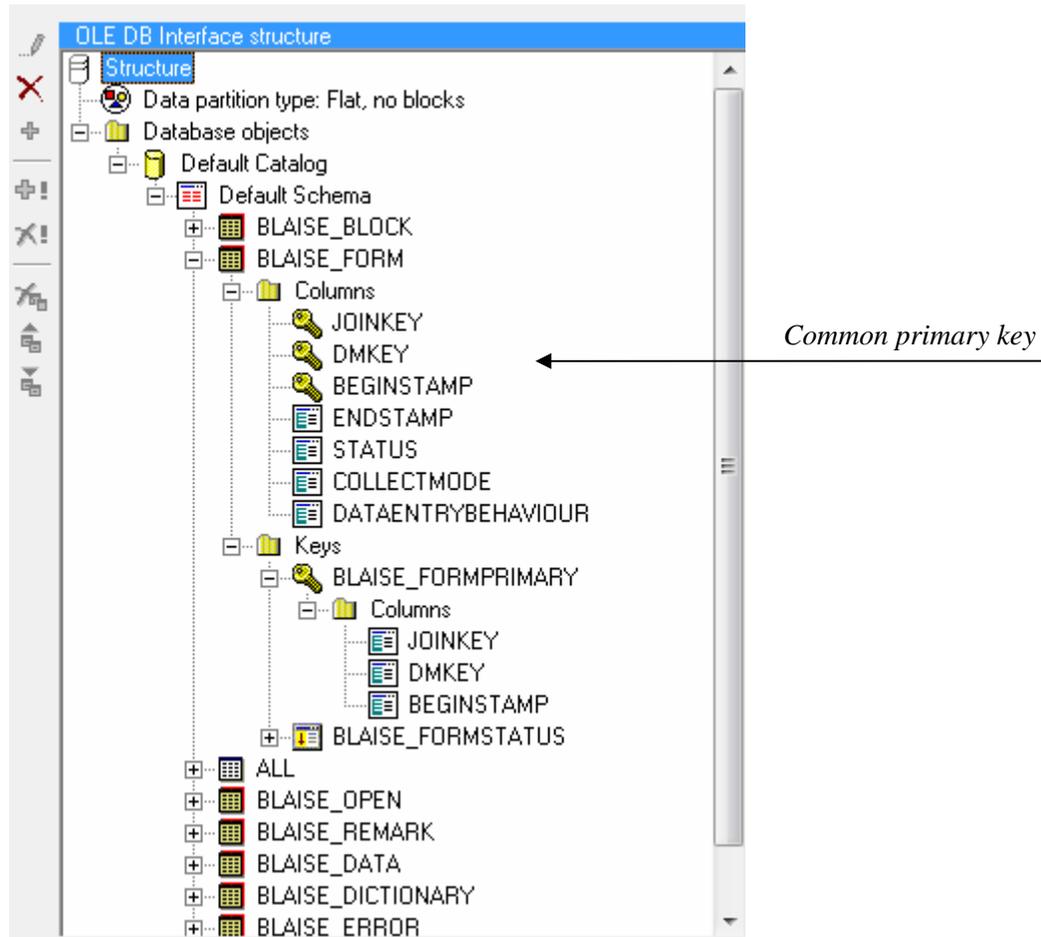


Figure 3.1 Structure of the generic Form information table

JOINKEY serves as a unique identifier for a case within Blaise. JOINKEYS will be automatically generated by the target database if a record does not exist already.

DMKEY is the key of the data model which belongs to the record data. Data stored with Blaise Datalink has always a link to its meta, because Blaise wants to know what the data represents. So we have no data without meta data!

Here is some more information about the data model key and why the key is important. Whenever Datalink opens a connection to the target database, a check occurs whether the data model that is being used is already registered in the Data Warehouse.

If the data model is not registered yet, then Blaise Datalink generates a DMKEY for the data model and also adds a new row with this new key to the *Dictionary Information* table. This is a fixed and predefined table which is included in all generic BOI files. The table stores all known data models along with some information, among others the paths of the BMI and BOI file which must be used.

The *Dictionary Information* table stores also the check sum of the data model. You can look at this check sum as a unique identifier for a version of a data model. If you change something in your data model, then the check sum will differ from the previous version of your data model. Blaise Datalink uses this check sum to distinguish between versions of data models. If a check sum differs from the check sum from an already stored data model, then a new DMKEY will be generated and as a result data will be stored along with the new meta information.

BEGINSTAMP is the begin time of the period of validity of a particular record. Whenever a record is stored for the first time then *BEGINSTAMP* will be filled with the time stamp of that moment. Begin stamps are generated automatically in the database. The main goal of the *BEGINSTAMP* column is that it can be used to distinguish between versions of records. If versioning is enabled then a new version of the record will be saved with a new begin stamp. You can read more about versioning in the next chapter.

Although generic BOI files use this common primary key in order to store data in a generic way, Blaise end users are not aware of this; it works completely transparent; users just need to define, as usual, primary and secondary keys in their Blaise data models.

3.2 Fixed table structures

Generic BOI files use also fixed and predefined table structures, while non-generic BOI files use tailor made table definitions. In the latter case only columns which are actually needed will be part of the table definitions. For example; if you have a data model with only integer fields and you choose the in depth data partition type then the resulting in depth data table will contain only a data column to store the integers. If you choose to create a generic BOI file then the table will have columns to store the other data types too. This is because other (future) data models might need these columns. By defining columns for each data type, regardless whether they are needed, the table can be shared between data models.

3.3 Use of maximal allowed column widths

Generic BOI files use maximum column widths supported by the particular database. Non-generic BOI files determine the column width which is needed to store the information. For example; if you have a data model with string fields and you choose the in depth data partition type then the resulting in depth data table will define a STRINGDATA column which has a width which corresponds to the largest string field size in your data model. If you choose to create a generic BOI file however, then the STRINGDATA column width will be set to the maximum column width supported by the database, again because future data models might require such a width.

4. Versioning

A new and interesting feature of Blaise 4.8 Datalink which we haven't told about yet, is versioning. Versioning can be used to store multiple versions of a Blaise record in the database. These versions can be retrieved on demand, for example with help from our new tool: Blaise OLE DB Data Center.

If you are using versioning then each record has a begin stamp which discriminates between the versions. Versioning can be switched on and off by altering the BOI runtime settings in OLE DB Workshop. Versioning can only be used with generic BOI files.

4.1 Versioning concepts

Our implementation of versioning works with so called time stamps. We have introduced two columns in order to enable versioning, namely:

BEGINSTAMP This column contains the begin time of the period of validity of a particular record. This value has to be unique in combination with JOINKEY and DMKEY.
ENDSTAMP This column contains the end time of the period of validity of a particular record.

These columns are defined as date time columns in the target database. Their precision is seconds for most databases. *BEGINSTAMP* is also part of the common primary key and stores the begin time of the period of validity of a record.

ENDSTAMP contains the end time of the period of validity of a record and has to be always greater than the *BEGINSTAMP* of that record. If *ENDSTAMP* is empty then the record is the actual record.

In its simplest form versioning works as follows. Whenever a record is written to the database then Blaise Datalink checks whether there is already an older version of the record. If this is the case then this older record will be closed by updating its *ENDSTAMP* column. Blaise Datalink will also insert a new row for the new version of the record. This row will have a new *BEGINSTAMP*, which is filled with the execution date and time. The *ENDSTAMP* column of this newly inserted row will be left empty.

When we take a closer look to versioning enabled records, we can differentiate between *actual* and *historical* records. *Actual* records are the records which contain the current values. Their *ENDSTAMP* column is empty. If a new version of a record becomes available then the *ENDSTAMP* column will be filled with the current timestamp. The

record becomes historic, because we have a newer one. *ENDSTAMP* contains the end time of the period of validity of the record.

4.2 Transparency

Versioning works completely transparent to the user, i.e. the user doesn't see whether versioning is enabled. He/she just uses the DEP, Manipula and Dataviewer as usual by using the keys defined in the Blaise data model. Despite the fact that there could be more versions of a record in the target database, only *actual* records will be used by the Blaise client tools.

4.3 Versioning example

Let's say we have a Blaise record and we haven't any versions of it yet in the target database. If we write this record to the database (we save the record in the DEP for example) then our database might look as follows:

	JOINKEY	DMKEY	BEGINSTAMP	ENDSTAMP	STAT...	COLLECTMODE	DATAENTRYBEHAVIOUR
1	997	1	2007-06-18 08:49:54.873	NULL	8	-1	0

We just have taken one table (Form information table) for this example, but all BOI involved tables will have the same primary key after the insertion of our record. The following important things happened behind the scene during the insert:

1. A *JOINKEY* has been generated for the case record. This is because we didn't have a record for this case yet. The *JOINKEY* serves as an identifier for a case in Blaise. *JOINKEYS* are generated automatically by the target database
2. You also see a *DMKEY* column. In this column you find the key of the data model that is being used. Data model keys will be generated at the moment that you open a Blaise Datalink for a Blaise data model for the first time.
3. *BEGINSTAMP* has also been generated by the database and contains the begin period of validity for the record.
4. *ENDSTAMP* is empty at the moment, because it is the only and thus the actual record. The other columns are not important for now.

Now let's add a new version of the record. This might be done as follows. Get the current record in the DEP, alter a value of a field and save the record. Here is our Form information table again:

	JOINKEY	DMKEY	BEGINSTAMP	ENDSTAMP	STAT...	COLLECTMODE	DATAENTRYBEHAVIOUR
1	997	1	2007-06-18 08:49:54.873	2007-07-17 12:27:39.520	8	-1	0
2	997	1	2007-07-17 12:27:39.520	NULL	1	-1	3

There are now two rows in the table; one row for the 'old' record and one for the new version. In the 'old' row *ENDSTAMP* has been filled with the date and time stamp of the executed write action. The same timestamp has been used to fill the *BEGINSTAMP* column of the new record. Please notice that not only the Form information table has been updated, but that all involved BOI tables have been updated according to this mechanism.

You can also see that the form status changed from 8 (= Unprocessed) to 1 (=Clean) and data entry behaviour changed from 0(=Unchecked editing) to 3(=Strict Interviewing). This is because the record was loaded and checked by the data entry program.

4.4 Versioning and write actions

As said, versioning can be switched on and off by altering the corresponding BOI runtime setting. This can be done in the Runtime settings tab within OLE DB Workshop.

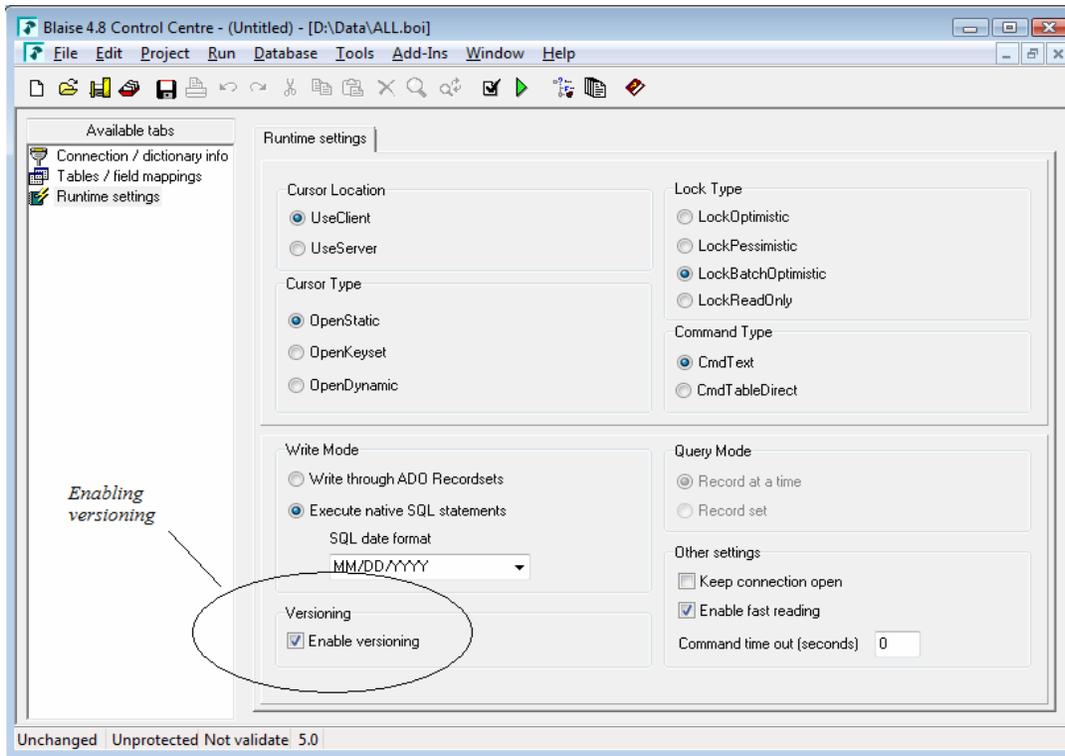


Figure 4.4.1 Versioning can be enabled by checking 'Enable versioning'.

Blaise supports several write actions and each of these actions has a different execution strategy. Here is a listing of these actions and the behavior in case we haven't enabled versioning:

- *WriteRecord*: writes a Blaise record to the database. If your data model has a primary key and a record with this primary key doesn't exist yet, then a new record will be created, otherwise the existing record will be overwritten with the new values. If you don't have a primary key then a new record will be added.
- *CreateRecord*: creates a new record in the database. If your data model has a primary key and a record with this primary key doesn't exist yet, then a new record will be created, otherwise an error will be raised, because you want to create a record which already exists. If you don't have a primary key then a new record will be added.
- *UpdateRecord*: updates an existing record in the database. If your data model has a primary key and a record with this primary key doesn't exist yet, then an error

will be raised, because you want to update a record which doesn't exist yet. If you don't have a primary key then an error will be raised.

As you see the working of these write actions is really straightforward. Now we have introduced versioning however, the situation becomes a little more complex. Here is a table which shows you the results of a write action in a particular situation:

Action / Setting / History	Versioning disabled		Versioning enabled	
	No record history present	Record history is present	No record history present	Record history is present
WriteRecord	Create new record BEGINSTAMP = current time and ENDSTAMP = empty	If there is an actual record (ENDSTAMP is empty), then update this record, otherwise create new record with BEGINSTAMP = current time and ENDSTAMP = empty	Create new record BEGINSTAMP = current time and ENDSTAMP = empty	If there is an actual record (ENDSTAMP is empty), then update ENDSTAMP of this record with current time. Also create a new record with BEGINSTAMP = current time and ENDSTAMP = empty
UpdateRecord	Error	If there is an actual record (ENDSTAMP is empty), then update this record, otherwise error	Error	If there is an actual record (ENDSTAMP is empty), then update ENDSTAMP of this record with current time. Also create a new record with BEGINSTAMP = current time and ENDSTAMP = empty
CreateRecord	Create new record BEGINSTAMP = current time and ENDSTAMP = empty	If there is an actual record (ENDSTAMP is empty), then error, otherwise create new record with BEGINSTAMP = current time and ENDSTAMP = empty	Create new record BEGINSTAMP = current time and ENDSTAMP = empty	If there is an actual record (ENDSTAMP is empty), then error, otherwise create new record with BEGINSTAMP = current time and ENDSTAMP = empty
DeleteRecord	Do nothing	If there is an actual record (ENDSTAMP is empty), then update ENDSTAMP of this record with current time.	Do nothing	If there is an actual record (ENDSTAMP is empty), then update ENDSTAMP of this record with current time..

Figure 4.4.2 Write actions and their result in a particular situation.

In general versioning works as follows:

The current actual record will become a historical record if a newer version of that record becomes available. In that case the old version gets an ENDSTAMP. The new version will be inserted with a new BEGINSTAMP and becomes the actual record. Its ENDSTAMP column will be left empty.

4.3 Versioning opportunities

If you enable versioning you will be able to see and analyze the history of Blaise records. With our new tool Blaise OLE DB Data Center you can analyze the difference between versions of records and see how and when changes to records have been applied.

This is nice, but working with versioning and especially with BEGIN- and ENDSTAMPS opens also opportunities which you might not have realized yet. One of the most important features is that we now are able to look at our survey data at a specific moment and in a particular time frame, i.e. you could ask the Blaise system to retrieve the situation of a survey at moment X and compare this to the situation at moment Y, i.e. we have the possibility now to analyze the survey progress and data quality.

4.4 Versioning requisites

Versioning can only be enabled with generic BOI files. You will also have to use a target database for which we have enabled versioning. This is because the target database must have enough support for date time columns in the way Blaise Datalink uses them. The database, for instance, must have the capability to generate JOINKEYS and BEGINSTAMPS automatically. Not all databases have support for those features. At the moment of writing we have enabled versioning for Oracle, Microsoft SQL Server and MS Access databases. We expect however to add more databases to that list in the near future.

5. Generic BOI files in practice

If you want to store the data of your survey in a relational database, you can use Blaise OLE DB Workshop to create a BOI file for your data model. Please read the Datalink chapter in the Blaise Online Assistant for a thorough explanation about creating BOI files and the available create options. In short you will have to do the following: First enter a connection string and Blaise dictionary in the Connection tab. Then go to the Tables / Field mappings tab and double-click the 'Create table definitions for dictionary' option. Now a 'Create table structure for dictionary' dialog pops up. If you want to create a generic BOI file then you must use the 'Advanced' tab.

In the Advanced tab you have two options:

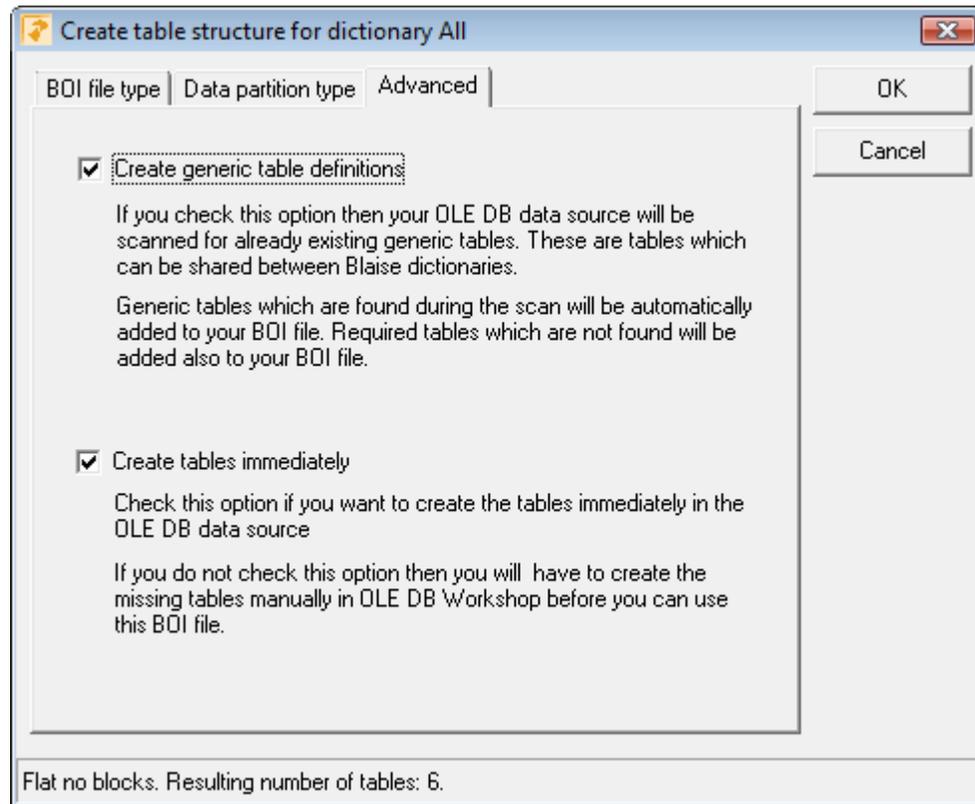


Figure 5.1 Advanced settings for generic BOI files

Check 'Create generic table definitions' if you want to create a generic BOI file for your data model. If you check the option then your OLE DB data source will be scanned for already existing generic tables. Found which are found are automatically added to your BOI file. Required tables which are not found will be added also. Note that the generic option is only enabled if your database supports versioning. See the specified requisites elsewhere in this paper.

After the generic BOI file has been created, OLE DB Workshop shows the table definitions that have been created.

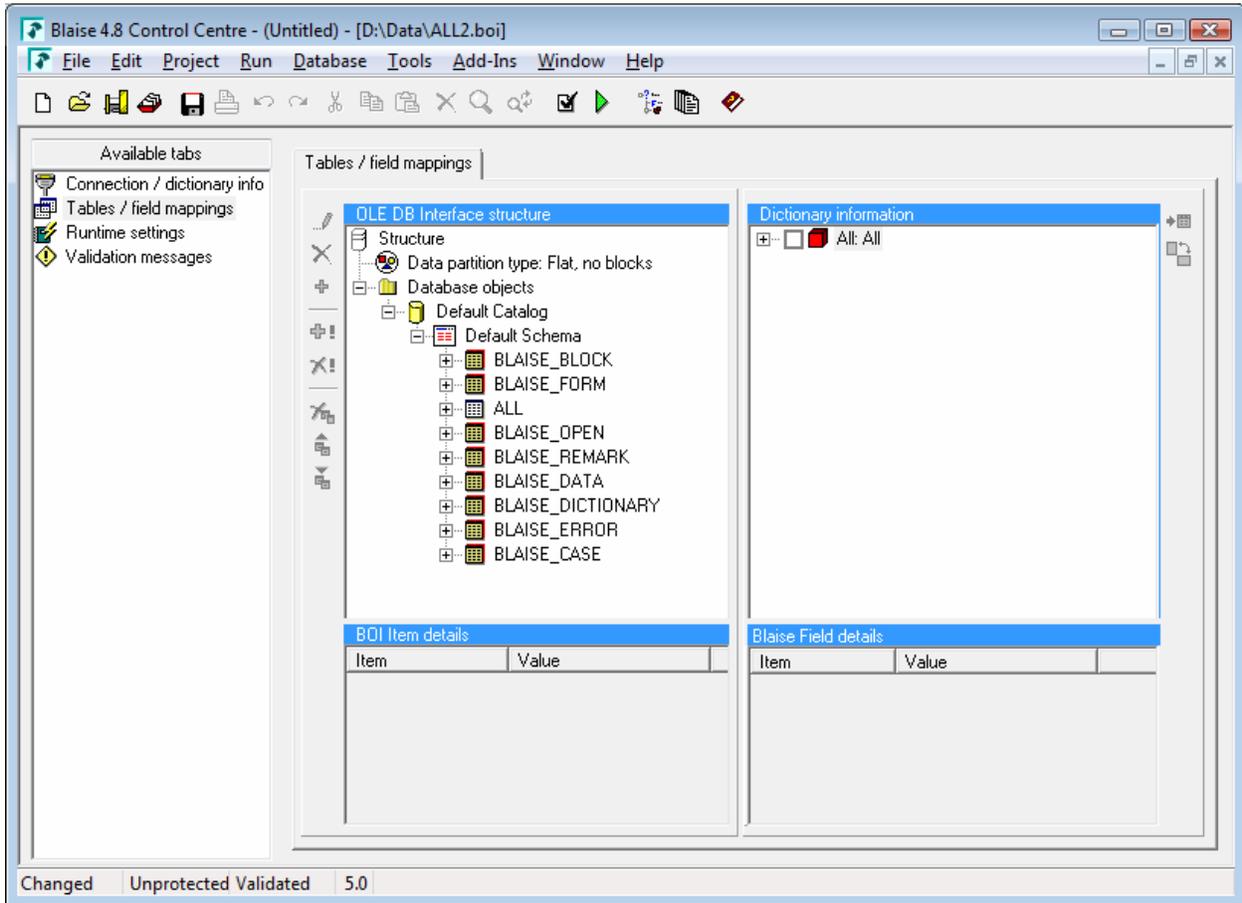


Figure 5.2.1 Generic table definitions displayed in OLE DB Workshop

We have a BOI file now with table definitions which can be used to store the data of the specified data model. This process has created a BOI file with logical table definitions; the tables themselves have not been created yet. The BOI file is a kind of blue print of what should be present in the database.

Be careful whenever you drop and create tables in the database!

Because we are working with a generic BOI files, it could be possible that some tables or even all the tables already exist in the database, because they were created by another generic BOI file earlier. This is because generic BOI files use the same table structures and, as a default, the same names in each generic BOI file. So you have to be careful with dropping and creating tables, because tables could exist already and contain data of other questionnaires!

To check whether a BOI file is ready to be used, you can validate the BOI file by choosing 'Project – Validate' from the menu. If everything is okay, then you will get the message 'OLE DB content is valid' otherwise errors / inconsistencies are listed in the 'Validation messages' tab. Missing tables will be listed there.

After validation you can see which tables are missing. You can create missing tables by selecting them in OLE DB Workshop and right clicking them. Choose 'Create table in OLE DB data source. Do this for all the tables which have not been created yet.

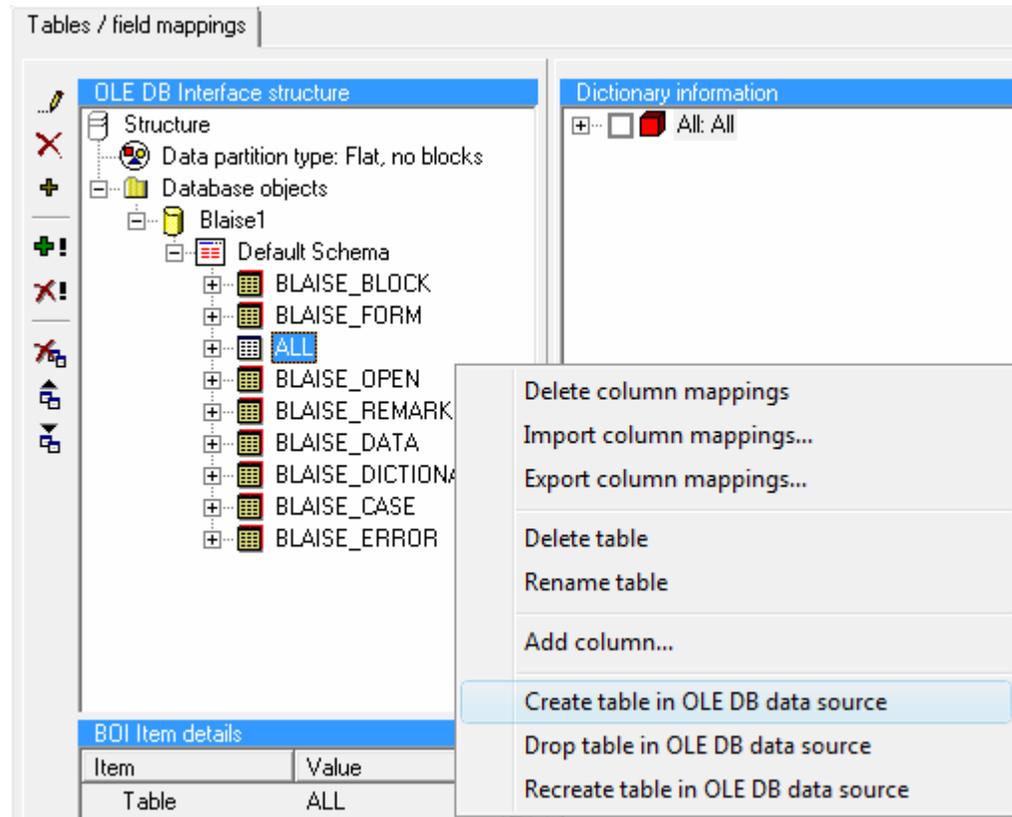


Figure 5.2.2 Create missing tables in the OLE DB data source

6. Blaise OLE DB Data Center

Generic BOI files offer many opportunities. If you store questionnaire data that belongs to different surveys in the same set of tables, you can speak of an input data store. Because questionnaire data is stored in a centralized way and as a result the available data can be overwhelming, we must have a way to look at and extract questionnaire data in an easy way. Also we might be interested in the progress of a survey. Blaise 4.8 Datalink supports versioning, so we must also be able to look at the history of records.

That is what the new Blaise OLE DB Data Center is all about. Its main purpose is to let Blaise users view and extract data in a uniform and easy way, without bothering about underlying table structures. The idea is that the view and extraction mechanism works exactly the same for all BOI files, so you have to learn it only once and then you can apply it to all BOI data.

I will explain the main features of this new tool below. Please notice that at the moment of writing Blaise OLE DB Data Center was still under development and that features might not be available in the final version and that screenshots may differ from the final version.

6.1 Data connections

Blaise OLE DB Data Center uses data connections in order to access the questionnaire data in a relational database. There are two types of data connections that can be made. The first type is an OLE DB connection string to a target database. The second type is a BOI file.

The difference between these two connection types is that in case of an OLE DB connection string the target databases will be scanned for registered dictionaries and accessible tables, while in case of a BOI file only BOI tables and its associated Blaise dictionary will be loaded into OLE DB Data Center.

Data connections are displayed in OLE DB Explorer which is the tree view on the left part of the screen.

OLE DB Explorer is the entry point of the application; you have to specify a data connection before you can do anything with OLE DB Data Center

In this example you see an instance of OLE DB Explorer with two connections: one to a BOI file and one with a connection string to a SQL server database.

Each connection shows the available Blaise dictionaries and accessible database tables.

Data connections are saved to file between sessions with OLE DB Data Center, i.e. you don't need to specify these connections again if you start up the application.

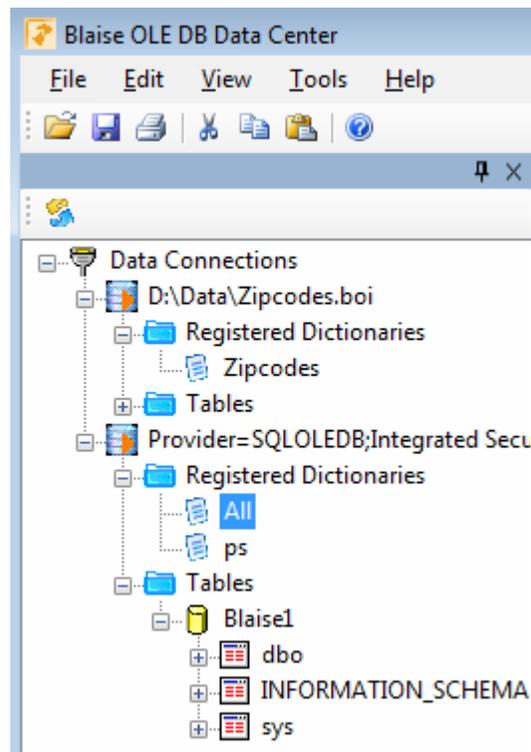


Figure 6.1 OLE DB Explorer in action.

6.2 Combining Meta and Data

If you are using Blaise Datalink then there must always be meta information available about the data that has been stored in the relational database. Datalink must always be able to access the Blaise dictionary that is associated with a particular BOI file. The guaranteed availability of meta is a big plus. Ask yourself: how often does it happen that data has been stored somewhere and no one knows what the data represent? Blaise Datalink simply requires meta in order to get the right data for each field in your Blaise data model out of the database.

How does Blaise Datalink knows which meta must be used?

The answer is simple: a Blaise OLE DB Interface (BOI) file contains a path and search path to the Blaise dictionary that must be used along with the data.

If you are using generic BOI files then the associated dictionaries are not only stored in the BOI files themselves, but also registered in a Dictionary table in the target database. This opens the possibility to provide a catalog function of registered dictionaries and BOI files which are used in an input data store.

Each dictionary has a data model key, which is also part of the common primary key of each database table. In this way Blaise Datalink is able to distinguish which data belongs to which data model. You can see this catalog function in OLE DB Explorer. The listed dictionaries in the Registered Dictionaries node have been retrieved from the Dictionary information table which is located in the relational database.

6.3 Viewing and extracting data

Whenever you open a data connection in OLE DB Data Center the Explorer searches for dictionaries which can be used.

Found dictionaries are listed under the Registered Dictionaries node, which becomes available after a data connection has been opened. If you select a dictionary then you can pop up a dictionary menu by clicking the right mouse button:

If you click View/Extract Data menu entry then a window is opened that can be used to view and extract questionnaire data.

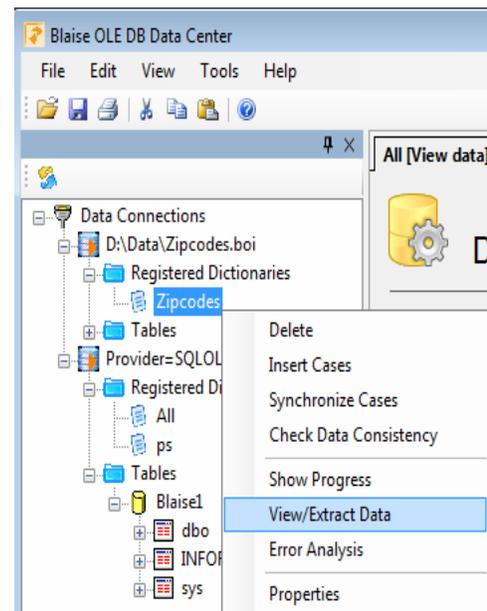


Figure 6.3.1 Dictionary menu

A nice thing about viewing and extracting data with OLE DB Data Center is that you only have to learn once how to specify what you want to see and extract and that it works for all questionnaires in the same way.

After you selected ‘View/Extract Data’ from the submenu, the screen looks like this:

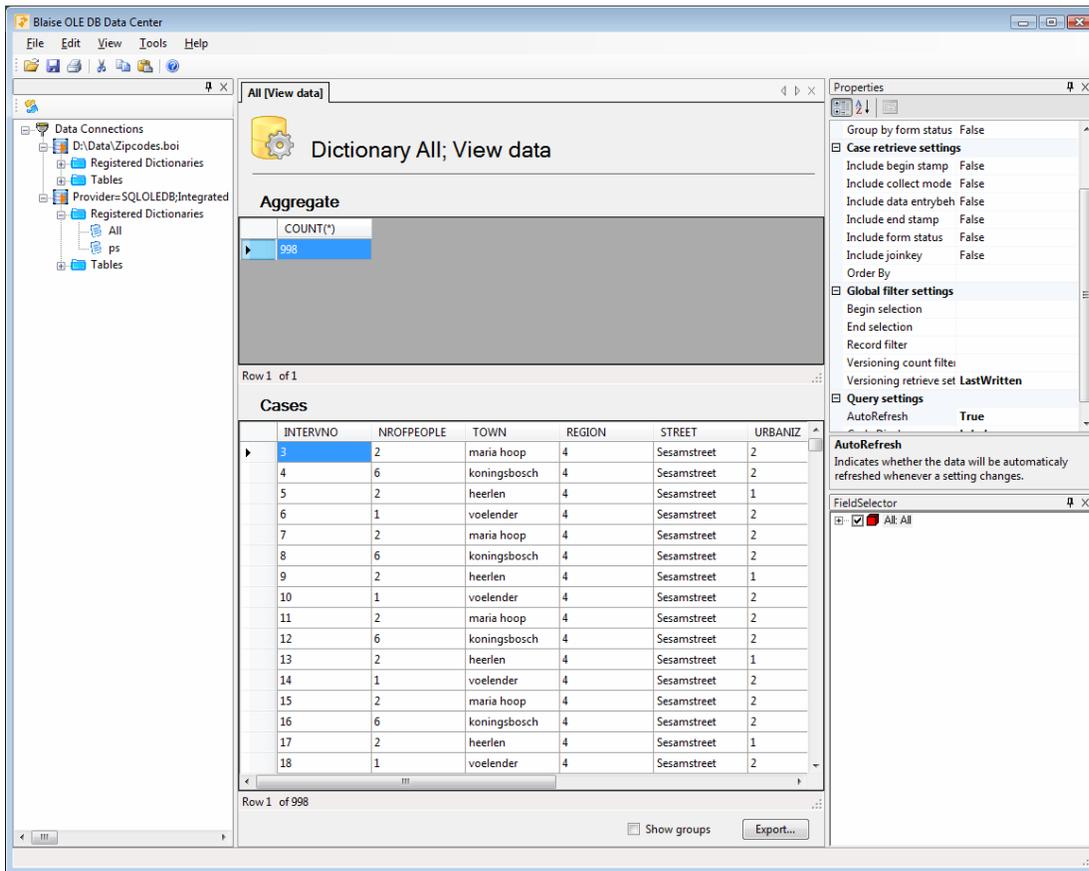


Figure 6.3.2 View/Extract Data: initial screen

The screen is now divided into four parts. On the left you see OLE DB Explorer. From here you can initiate other actions at any moment. In the middle we have a ‘View data’ document in which data is displayed and on the right you can find Properties and a Field Selector. You can use the latter two to specify which data you want to retrieve from the target database.

The ‘View data’ document, which is the active document right now, is divided into an *Aggregate* part and a *Cases* part. These two parts work together. *Aggregate* contains, as the name already indicates, aggregated data. The *Cases* part contains Blaise cases which are involved in the selected row in the *Aggregate* section.

In this screenshot the count (number of records) of data model All is displayed in the upper section, while the cases sections contains the cases which are involved in this count.

How do the available screen parts interact with each other?

You can use the Properties window on the right to specify retrieve settings. This Properties window contains retrieve settings in case you are viewing and extracting data. Examples of retrieve settings are field and record filters, but also group by settings.

The available retrieve settings can be divided into four parts.

Aggregate settings are settings that work on the aggregate part of the screen. Here you can specify how data will be grouped. Data can be grouped by form status, data entry behavior, data collect mode and by field data.

Case settings can be used to influence which case information will be retrieved in the bottom half of the data document. You can include join keys, begin- and end stamps, form status, et cetera. These settings work in conjunction with the field selector, which you can use to specify which field data should be retrieved.

Global filter settings work on both aggregates and cases. These settings can be used specify a begin and end time and a record filter. There are also some settings which are related to versioning.

Finally there are also some *general* settings.

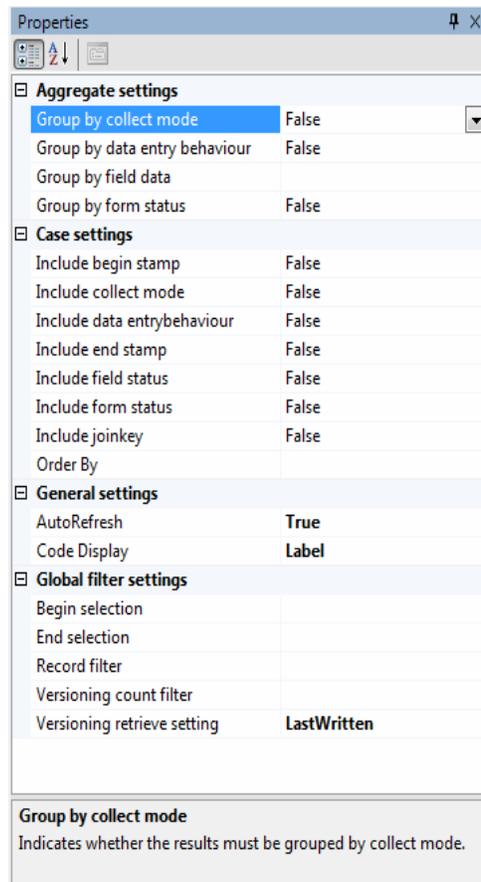


Figure 6.3.3 Available settings

By using retrieve settings in combination with field selector you can retrieve data from the database. Once the information is displayed in the *Cases* part of the screen, you can export the data to other applications and file formats. Blaise OLE DB Data Center supports several output formats. You can, for instance, copy the data and paste it in Microsoft Excel or SPSS or export the data to a text file.

These export facilities are still under development at the moment of writing, so I can't go into much detail about it yet, but it should be more concrete during the time that this paper is presented.

6.4 Administrative tasks

Blaise OLE DB Data Center has two execution modes: administrator and user. If you are logged on as an administrator then administrative tasks will be available next to the default features. The extra mode has been introduced because some people must be able to administrate the surveys in the IDW.

Tasks you can think of are the following

- Adding and deleting surveys
- Inserting and deleting cases
- Importing data
- Synchronizing cases
- Performing data validation checks

At the moment of writing this tool is still under development, so features might not be available in the final version.

7. Conclusion

Blaise 4.8 Datalink has a number of interesting new features. You can use generic BOI files to setup an Input Data Warehouse in which the data of multiple Blaise surveys is stored. Blaise end users can use Blaise OLE DB Data Center to view and extract the data they want in an easy and uniform way.

We also have seen another new feature of Blaise 4.8 Datalink which is versioning. Versioning makes it possible to store the history of a Blaise record. This gives us the possibility to view the changes made to a record in time but also to look at a survey within a particular time frame. It is possible to analyze the progress of the survey and also the progress in data quality.

8. Appendix Predefined generic table structures

Generic BOI files use predefined fixed table structures as said earlier. In this section you find an overview of these tables and their structure. These table structures are defined in any generic BOI file.

8.1 Dictionary information table

The Dictionary Information table stores information about the dictionaries which are used in the IDW.

The default name of this table is `BLAISE_DICTIONARY`. Its structure looks like this:

DMKEY	Data model key
DATAMODELNAME	Name of the data model
CHECKSUM	Check sum of the data model
BMI	Path to the associated dictionary file
BOI	Path to the associated BOI file
SEARCHPATH	Dictionary search path
ADDED	Date and time of adding
COLLECTMODES	Supported collect modes for this dictionary. Not implemented yet

8.2 Case information table

The Case Information table is used to store the `JOINKEY`s of a survey. If a new, non existing case is written to the database then a new `JOINKEY` is generated and stored into this table.

The default name of this table is `BLAISE_CASE`. Its structure looks like this:

JOINKEY	Unique integer which identifies a case.
DMKEY	Data model key
COLLECTMODE	Default collect mode for this case. Collect mode has not been implemented yet.

This table can possible be extended by a user defined data model. We were exploring this possibility at the moment of writing this paper.

8.3 Form information table

The Form Information table is used to store the status information about a form. The table contains also information about the collection mode of the form and the data entry behavior which was being used at the moment the record was written.

The default name of this table is `BLAISE_FORM`. Its structure looks like this:

JOINKEY	Unique integer which identifies a case.
DMKEY	Data model key
BEGINSTAMP	Begin time of the period of validity of a particular record. This value has to be unique in combination with <code>JOINKEY</code> and <code>DMKEY</code> . This column is used for versioning.
ENDSTAMP	End time of the period of validity of a particular record. If this column has an empty value than it is the actual record.

STATUS	Form status of the record.
COLLECTMODE	Collect mode of the record. At the moment of writing this was not implemented
DATAENTRYBEHAVIOUR	Data entry behavior of the record.

8.4 Block information table

The Block Information table is used to store the block check status and error information. This is information that Blaise needs to set suppressed errors et cetera.

The default name of this table is BLAISE_BLOCK. Its structure looks like this:

JOINKEY	Unique integer which identifies a case.
DMKEY	Data model key
BEGINSTAMP	Begin time of the period of validity of a particular record. This value has to be unique in combination with JOINKEY and DMKEY. This column is used for versioning.
ENDSTAMP	End time of the period of validity of a particular record. If this column has an empty value than it is the actual record.
BLOCKID	ID of the block for which the error information is stored.
STATUS	Status of the block.
STREAMDATA	Stream that contains the data of the block.

8.5 In depth data / status table

The In Depth data / status table will be used in case you select one of the two in depth data partition types or if your data model has don't know or refusal fields.

The default name of this table is BLAISE_DATA. Its structure looks like this:

JOINKEY	Unique integer which identifies a case.
DMKEY	Data model key
BEGINSTAMP	Begin time of the period of validity of a particular record. This value has to be unique in combination with JOINKEY and DMKEY. This column is used for versioning.
ENDSTAMP	End time of the period of validity of a particular record. If this column has an empty value than it is the actual record.
FIELDID	ID of the field to which the data belongs.
STATUS	Field status of the field
STRINGDATA	Field values stored as text (data partition type in depth text) or answers to fields with string data type (data partition type in depth)
INTEGERDATA	Answers to fields with integer and enumeration data type (data partition type in depth)
FLOATDATA	Answers to fields with real data type (data partition type in depth)
DATETIMEDATA	Answers to fields with date and time data type (data partition type in depth)

8.6 Remark table

Remarks to questions are written to this table.

The default name of this table is **BLAISE_REMARK**. Its structure looks like this:

JOINKEY	Unique integer which identifies a case.
DMKEY	Data model key
BEGINSTAMP	Begin time of the period of validity of a particular record. This value has to be unique in combination with JOINKEY and DMKEY. This column is used for versioning.
ENDSTAMP	End time of the period of validity of a particular record. If this column has an empty value than it is the actual record.
FIELDID	ID of the field for which the remark has been created.
REMARKTEXT	Text of the remark.

8.7 Open table

This table stores answers to open (memo) fields. Only open fields which have a non-empty answer text will be stored.

The default name of this table is **BLAISE_OPEN**. Its structure looks like this:

JOINKEY	Unique integer which identifies a case.
DMKEY	Data model key
BEGINSTAMP	Begin time of the period of validity of a particular record. This value has to be unique in combination with JOINKEY and DMKEY. This column is used for versioning.
ENDSTAMP	End time of the period of validity of a particular record. If this column has an empty value than it is the actual record.
FIELDID	ID of the open field.
STATUS	Field status of the open field
OPENTEXT	Answer text

	unique in combination with JOINKEY and DMKEY. This column is used for versioning.
ENDSTAMP	End time of the period of validity of a particular record. If this column has an empty value than it is the actual record.
FIELDID	ID of the field for which the remark has been created.
REMARKTEXT	Text of the remark.

8.7 Open table

This table stores answers to open (memo) fields. Only open fields which have a non-empty answer text will be stored.

The default name of this table is `BLAISE_OPEN`. Its structure looks like this:

JOINKEY	Unique integer which identifies a case.
DMKEY	Data model key
BEGINSTAMP	Begin time of the period of validity of a particular record. This value has to be unique in combination with JOINKEY and DMKEY. This column is used for versioning.
ENDSTAMP	End time of the period of validity of a particular record. If this column has an empty value than it is the actual record.
FIELDID	ID of the open field.
STATUS	Field status of the open field
OPENTEXT	Answer text

Methods for Simultaneous Meta & Data manipulation in Blaise

Marien Lina, Statistics Netherlands

1. Introduction

From the beginning of the Blaise development, there is an old saying going “no data without meta data”. If you have data without a data description then it is obvious that you in fact have nothing at all. Without a meta data file Blaise does not know what the structure and the meaning of the data is and you will not be able to use the data within Blaise.

Until relatively recently the meta data of Blaise and the data of Blaise were in principle two separate worlds. Although Blaise can handle data only when the meta data is available, the handling of meta data and of data by the user have been separated in different parts of the Blaise system. The Blaise data model language and the Manipula language are primarily equipped for handling of the data in Blaise and the Cameleon language is primarily equipped for handling of the meta data in Blaise.

With the release of the Blaise API in 2001 it became possible to combine meta data and data in one application. This offered many new (and often unforeseen) possibilities. There is one ‘major’ drawback: you need IT skills and access to IT tools to be able to use the Blaise API.

Lately with the release of Blaise 4.8 a new possibility has become available to combine meta data and data. By using Manipula, Blaise 4.8 now also offers in the basic system a way to combine meta data and data. This paper describes the background of combining meta data and data in Blaise and how this can be used.

2. The Blaise and the Manipula language

The Blaise and Manipula languages have primarily been designed to handle data rather than meta data.

The Blaise language describes the structure and the relations in data. You describe blocks, types, fields, routing, relations, computations, external lookups etcetera with as main purpose to collect and validate data based on the specification. The Blaise language in itself has no way to access the meta definition actively during the collecting and validation process. In other words: despite the fact that field names and question texts are visible in the DEP on your data entry screen, there is no possibility to retrieve meta information from the meta file and use it in the rules. For example, you cannot ask for the field name or the field type of a certain field and assign it to a string field.

The Manipula language also aims at data handling. The language is great for many things like import/export, selections, sorting etcetera but within Manipula it is not possible to actively use the meta specification of the data of the files it is processing. Because of that it is for instance not possible to write a generic Manipula setup that produces a report that gives a list of answered questions showing both the question text and the given answer.

3. The Cameleon language

To read meta data from a Blaise BMI file and to write it in another format, something else than the Blaise and Manipula language was needed. In Blaise 2 the setup generator was introduced, replaced in later Blaise versions by Cameleon.

Cameleon uses setups specified in the Cameleon language. It enables you to present meta data in another format. This may be for documentation or for alternative data descriptions, suitable for converting Blaise data to external statistical packages and databases. Cameleon retrieves meta information from the BMI, manipulates its appearance to a specific form and puts the result in a text output file. The Cameleon language is not simple and it requires some skill to get the most out of Cameleon. Luckily numerous Cameleon setups are shipped with Blaise. They serve as examples for creating new setups.

There is an analogy between a Cameleon setup and a style sheet. Both are used to reshape in a generic way that what they process. A Cameleon setup reshapes meta data and a style sheet reshapes XML documents. You can read more on this topic in the paper “On the use of XML in the Blaise environment” by Jelke Bethlehem and Lon Hofman, proceedings of the International Blaise Users Conference in Kinsale in May 2000.

4. Combining data and meta data: the Blaise API

With Blaise 4.5 (2001) the Blaise Application Programming Interface (API) has become available. It is shipped as part of a separately licensed product called the Blaise Component Pack (BCP). The Blaise API enables you to address the data and meta in one application. The advantage is clear: with the Blaise API you can combine the Manipula and Cameleon functionality into one application. You can use the Blaise API in any programming language that supports COM technology (the Component Object Model from Microsoft).

Both the DEP and Manipula can easily interact with components using the Blaise API. Within the DEP these components can be called via the menu, via the alien procedure mechanism or via the alien router mechanism. The components can also be linked to a user defined type.

In short, the Blaise API makes it possible to create your own applications and address data and meta data in the same application. Based on the API it is in principle possible to develop applications that replace systems that combine Manipula and Cameleon applications.

Note that using the API requires knowledge of modern programming languages, it requires programming skills, and it requires access to advanced system development tools. This may be a barrier for the Blaise programmer or survey designer.

5. Introduction of API functionality in Manipula.

With the API in mind two developments took place for Blaise 4.8. Firstly, meta data access functionality has been added to Manipula, and secondly, the communication between the rules and Manipula has been made possible, thus

offering Manipula procedures as an alternative for DLL calls and ActiveX calls in the rules.

There were two reasons for doing this. The main reason is related to the requirements imposed on the development of Basil, a new tool in Blaise 4.8 that allows for 'tailored user interfaces' in a CASI setting. Basil applications should be able to run on machines that have no Blaise components installed (so no API access) but they should still be able to do most things that are only possible when using the Blaise components. The second reason is to make it possible for qualified Blaise programmers to extend the system without having to go through programming using IT tools.

Most requirements for Basil could be met only when Manipula and Basil would work closely together and when Manipula was able to access the meta data of the data model. The generic solution that was implemented for Blaise 4.8 is based on a flexible, event driven connection between Basil, the rules and Manipula setups.

6. Calling Manipula procedures in the rules

Running Manipula from within the DEP was first made available in Blaise 4.7. From within the menu of the DEP you are allowed to call a Manipula setup and in that setup you can have access to a copy of the form that is being handled by the DEP. To get access to that form, you need to use the file setting `INTERCHANGE=TRANSIT`.

In Blaise 4.8 the implementation of `INTERCHANGE` has been extended in such a way that you now can have access to the actual form that is in the DEP session. To achieve this you need to use the file setting `INTERCHANG=SHARED`. Accessing the actual form instead of a copy is not only faster but makes it possible to give you access to much more information like the actual errors present in the form and routing status information without having to execute the rules again.

Blaise 4.8 also makes it possible that at all places where you are allowed to call a method of a component you now also can call a procedure in a Manipula setup. This makes it possible to call Manipula from within the rules by using alien procedures and alien routers and this makes it possible to use features in the rules that were not available previously.

Below is an example of a small Blaise data model ASurvey that uses an alien Manipula procedure ComputeC.

```
DATAMODEL ASurvey

PROCEDURE ComputeSideC
PARAMETERS
  IMPORT A, B: REAL
  EXPORT C: REAL
ALIEN('MySetup.msu', 'ComputeC')
ENDPROCEDURE

FIELDS
  SideA, SideB "Enter value": 1..20
  SideC "Side C will be:": 1.41..28.28
```

```

RULES
  SideA  SideB
  IF (SideA>0) AND (SideB>0) THEN
    ComputeSideC(SideA, SideB, SideC)
  ENDIF
  SideC.show
ENDMODEL

```

When values have been entered for SideA and SideB, then the alien procedure ComputeSideC is called and that procedure then calls the procedure ComputeC in the Manipula setup *MySetup*. It computes the value for C and returns it to the field SideC.

```

PROCESS MySetup

PROCEDURE ComputeC
PARAMETERS
  IMPORT A, B: REAL
  EXPORT C: REAL
INSTRUCTIONS
  C:= SQRT(A*A + B*B)
ENDPROCEDURE

```

It is just a simple 4.8 example to show that you can address Manipula procedures in the rules. Within those procedures you have access to all the richness of Manipula thus allowing to write to files, display a tailored dialog box, call other applications like another data entry session etcetera. This can now all be done using the Blaise basic system tools.

7. Retrieving meta data in Manipula

For meta data handling, a relevant new extension in Blaise 4.8 is the introduction of methods to retrieve meta data with Manipula. Like the Blaise API, a Manipula setup can now access the meta data of a file that it is using. You can access all meta data for the fields and types, like field names, question texts etcetera, but you can also access meta data like error information and routing status.

In the Manipula setup you can loop over all fields and blocks (recursion has been introduced in Manipula to make this easier). For every element (for example a field) in the loop you could print the field label, the question text, etcetera. In the Blaise basic system this could until now only be done by using Cameleon. In fact, many (although not all) things you can do in the Cameleon language are now also possible in a Manipula setup in Blaise 4.8.

An example of a meta producing Manipula setup is included in the Blaise 4.8 samples folder *demo48\ManipulaMeta*. The main setup of this example is called *producemeta.man*. The snippet below shows a part of the source code:

```

WriteLine('Field Size:' + InputFile1.GETFIELDINFO(MyFieldName, 'SIZE'))

WriteLine('Attributes:' + InputFile1.GETFIELDINFO(MyFieldName, 'ATTRIBUTES'))

WriteLine('Field Type Name:' +
  InputFile1.GETFIELDINFO(MyFieldName, 'FIELDTYPENAME'))

WriteLine('Local Type Name: ' +
  InputFile1.GETFIELDINFO(MyFieldName, 'LOCALFIELDTYPENAME'))

```

The setup gets meta information from the BMI and writes it to an output file. The file method `GETFIELDINFO` has various keywords that can be passed as a string parameter, like `'size'` that returns the size of the field. Note that validity of those keywords is checked at run time. So beware of typos! There are a number of other methods available for retrieving meta information like `GETMETAINFO` and `GETTYPEINFO`. These methods open the way to create integrated systems in Manipula that used to be a combination of Cameleon and Manipula setups. Manipula setups could eventually replace Cameleon setups.

8. Accessing meta information in the rules

By combining the access to meta information within Manipula and the use of Manipula procedures within the rules, the meta data becomes also accessible within the rules. The example below shows how to get the name of an enumeration category and assign it to a text field.

First the Manipula procedure:

```

PROCESS GetMeta
USES MyMeta 'GetMeta'

TEMPORARYFILE MyData:MyMeta

PROCEDURE GetCategoryName
PARAMETERS
  IMPORT FieldValue: INTEGER
  IMPORT TypeName: STRING
  EXPORT CategoryStr: STRING
AUXFIELDS
  Indx: INTEGER
INSTRUCTIONS
  FOR Indx:= 1 to VAL(MyData.GETTYPEINFO(TypeName, 'CATEGORIES.COUNT')) DO
  IF
MyData.GETTYPEINFO(TypeName, 'CATEGORIES['+STR(Indx)+'].CODE')=STR(FieldValue)
THEN
  CategoryStr:= MyData.GETTYPEINFO(TypeName, 'CATEGORIES['+STR(Indx)+'].NAME')
  EXITFOR
  ENDF
ENDDO
ENDPROCEDURE

```

The procedure `GetCategoryName` in the Manipula setup receives the value of a field in the data model via the import parameter `FieldValue`. The value of the import parameter `TypeName` is used to access the meta information that belongs to the temporary file handle `MyData` using the method `GETTYPEINFO`. This method is used to determine the number of available categories, it loops over all available categories until the category code (again using the `GETTYPEINFO` method) matches the value of the import parameter `FieldValue`. The corresponding category name is returned to the rules using the export parameter `CategoryStr`.

The procedure `GetCategoryName` can be activated via the rules in a Blaise data model as an alien procedure.

```

DATAMODEL GetMeta

TYPE
  THP = (Man, Woman, Child, Otherwise)

PROCEDURE GetCatName
PARAMETERS
  IMPORT FieldValue: INTEGER
  IMPORT TypeName: STRING

```

```

    EXPORT CategoryStr: STRING
ALIEN('GetMeta.msu','GetCategoryName')
ENDPROCEDURE

FIELDS
    HousePos "Position in the household": THP
    CatString: STRING
RULES
    HousePos
    GetCatName(ORD(HousePos),'THP',CatString)
    CatString.SHOW
ENDMODEL

```

The example requires that the BMI used in the Manipula setup is the same as the data model that calls the Manipula setup. So the Manipula script needs to be prepared each time the data model structure changes. Blaise 4.8 also makes it possible to pass the BMI file as parameter to the Manipula setup. This allows you to re-use the prepared Manipula setup for any data model you have. It only requires two simple changes to the example. In the Manipula setup the meta identifier needs to be defined as a variable:

```

USES Meta (VAR)

```

And in the data model the alien call needs to be changes: it now needs to pass the name of the BMI file to the setup also. This looks as follows:

```

ALIEN('GetMeta.msu /KMyMeta=$dictionaryname','RetrieveCodeName')

```

At runtime `$dictionaryname` will be replaced by the name of the BMI file and by using the `/K` command line option of Manipula it is then passed to the setup linking it to the meta identifier `MyMeta`. Note that when you pass the BMI file as a parameter to the Manipula setup certain restrictions are imposed on the setup. You are for instance not allowed to access a field of a field directly but you will need to use the methods `GETVALUE` and `PUTVALUE` instead. The restrictions are checked during the preparation of the Manipula setup.

In the Manipula script you can also get access to all the data in the form that is in the DEP session that executed the rules. You have to use the `INTERCHANGE` setting to achieve that:

```

TEMPORARYFILE Data:Meta
SETTINGS
    INTERCHANGE=SHARED

```

9. Conclusions

While in older versions of Blaise meta data and data have been strictly separated in different parts of the system, new developments in Blaise make it possible to combine the retrieval of meta data and data in Manipula, and via the alien procedure also in the rules of a data model. This opens many new possibilities in the Blaise basic system for instance to create systems in Manipula that process meta data, without having to use Cameleon.

Blaise and Xml: Experiences Of An Outsourcing Project¹

Leif Bochis Madsen, Statistics Denmark

Abstract

Xml is a standard for the representation of tree structures with decorated nodes and today it has become a common for exchanging information in a platform-independent way.

Because of their hierarchical nature Blaise data and meta data are quite easy to convert into xml, however, various demands may influence the way this should be done – among them conformance to standards requirements and, of course, the actual application.

Statistics Denmark has recently been through a process of outsourcing telephone as well as personal interviewing for the Labour Force Survey while post processing of forms including coding still should be carried out in-office.

Among the tasks was definition of a data exchange format using xml for the exchange of questionnaire data and development of tools for automatic generation of e.g. xml schemas to support validation of xml documents and for conversion between Blaise and xml.

The paper summarizes some experiences and challenges from this project.

1. Introduction

Early in 2006 it was decided that the interviewing for the Danish Labour Force Survey should be outsourced from January 2007.

During the spring and summer of 2006 Statistics Denmark prepared for a competitive tendering and in the autumn a contract was signed with a consortium of organizations on the interviewer market (hereafter ‘The Interview Consortium’).

Meanwhile, the development of the various artefacts needed to support the process began in order to be ready for a pilot study in November 2006 and for production in January 2007.

¹ The opinions and assessments stated in this article are those of the author and do not necessarily reflect opinions and assessments of Statistics Denmark.

1.1 Requirements

Essential for the architecture was the decision to exchange data in xml format. This was a demand published in the material for the competitive tendering, and the process of defining an exchange format started in the early summer of 2006.

Post processing of forms should be carried out in-office and it was decided to reuse the already developed Blaise instruments for this purpose. In order to save time it was then an obvious choice to use the previously used Blaise Interview instrument as the basis of developing an interview test instrument and as the metadata base on which we could automatically generate proper xml definitions. Experimental work had already been done concerning the representation of Blaise data in xml, but these experiments had never been used for practical work. Now, these efforts showed to be a firm base for the contemporary needs.

The software that Statistics Denmark needed to develop in order to support the new data capture process thus consisted of the following elements:

- A Cameleon script for the generation of an xml schema
- A VB program for changing the character set (as Cameleon is not capable of writing data in UTF-8)
- A VB program for extracting data into xml format (for testing the application)
- A set of further VB-programs /Cameleon scripts for converting the metadata into xml
- A C# program for loading xml data into Blaise
- A java script for changing the xml into a format suitable for reading into Blaise (splitting the data into different versions and changing some inapplicable codes into applicable ones)
- A Maniplus program for managing the process

The C# program and the java script were quite new artefacts while the rest all could be written as modifications of previous artefacts.

Further requirements for the project comprised an effort to conform to standards for data exchange set up by the Danish Ministry of Science, Technology and Innovation – the so-called OIO standards.

These standards are not mandatory for a “closed shop” data exchange, but reuse of elements already defined by other organizations and a general policy of Statistics Denmark to conform to the standards implies the use anyway.

We shall briefly describe the purpose and characteristics of these standards.

Also, the Blaise 4.8 xml export was yet to come, so we had to invent this wheel, too. Through a case study we shall take a look at some of the differences and similarities between Blaise 4.8 xml format and the xml exchange format for this project.

1.2 OIO – Public Information Online

The purpose of the Danish Government OIO (Danish acronym for “Public Information Online”) initiative is to improve standardization and data exchange.

As stated on the project homepage:

“The vision is that the Danish OIOXML project will:

- *Improve the exchange of data both internal in the public sector and between the public and private sector.*
- *Improve the processing of data, and make easy access to already collected data and the re-use of these.*
- *Make it easier to implement E-services*

To realise this vision two initiatives are initiated: The Infostructurebase and the standardising work “²

The Infostructurebase comprises among other things a set of definitions of public information in the shape of xml schemas. It is an obligation for a public organization responsible for some public information to maintain the xml schemas in the Infostructurebase and to provide the information available as xml data conforming to these definitions. Organizations comprise government agencies, municipal authorities, private organizations etc.

Among the standardization efforts are a restriction of how xml schemas should be designed:

“XML schema definitions

- *Write the tag names in English; the names should be indicative of the purpose. Danish descriptions should be schema definitions as alternative definitions.*
- *Standardise the naming of central elements. Those are e.g. primarily the central registers, in which it is important to establish the naming of central elements such as First name, Last name, Address, as these definitions are used across the boundaries of public authorities. Whether or not it is possible to use PersonName and other standards of XML.org should be investigated.*
- *Write the tag names with the following notation: "UpperCamelCase", i.e. initial capital letter, the rest lower case. Compounds such as Community Number should be written as "CommunityNumber".*

² From the OIO website

- *Fundamental attributes should be established in order to be able to make external references to the relevant registers.*
- *UTF-8 is to be used as the standard character set.*
- *If there are homogeneous constructions in more places, these should be made as independent definitions.*
- *Elements are to be documented in schema definitions, so that every field is well defined.*
- *Make strong data types. Ex.: For a date the data type DATE should be used in schema definitions.”³*

As mentioned in the introduction it is not mandatory to conform to this standard for peer-to-peer systems like the one developed for the exchange of data between Statistics Denmark and the Interview Consortium. It is, however, a general policy of Statistics Denmark to do it so far it is convenient, and some of the information used for the LFS actually stems from sources belonging to this scheme.

Therefore, as it was not possible to ignore the OIO we had to cope with it.

2. Architecture of the LFS data capture system

The capture of data for the new LFS now comprises the following work steps:

Each quarter a revision of the LFS questionnaire is carried out, for the most resulting in a revised version of the questionnaire.

The design of the questionnaire is carried out in the LFS section at Statistics DK and implemented in Blaise for testing purposes. At the same time a post editing tool is developed from the same Blaise source.

A description of the questionnaire along with an xml schema defining the interchange format is mailed to the Consortium for their implementation.

Respondent data are sent to the Consortium once a month in xml format.

Daily, an xml file is uploaded to Statistics Denmark containing the completed interviews from the previous day along with non-response and expired interviews.

The data from this xml file is loaded into a Blaise database for post editing.

2.1 Design of the questionnaire

Although it should not be used for actual interviewing, a questionnaire for each version of the survey is implemented as a Blaise instrument for the purpose of testing the filters.

³ Ibid.

By the use of prepare directives it has been quite easy to augment the source code of the interview instrument with source code for the data editing instrument.

As roughly 99.9 percent of the interview instrument may be reused for the data editing instrument and the interview instrument might comprise some 95 percent of the data editing instrument this approach has proven efficient.

Thus, by the end of the design and test phase, everything is prepared for receiving the data from the Interview Consortium.

2.2 Interchange format

From the Blaise interview instrument it is possible to extract the metadata by means of Cameleon or the Blaise API. Therefore, we developed a Cameleon script that could automatically create an xml schema to be used by the Interview Consortium for validation before sending the data. Also, some other documentation of the questionnaire is produced by Cameleon scripts and some VB programs using the Blaise API. This documentation is then sent to the Interview Consortium for their preparation of the questionnaire.

2.3 Overall architecture of the Data Capture Process

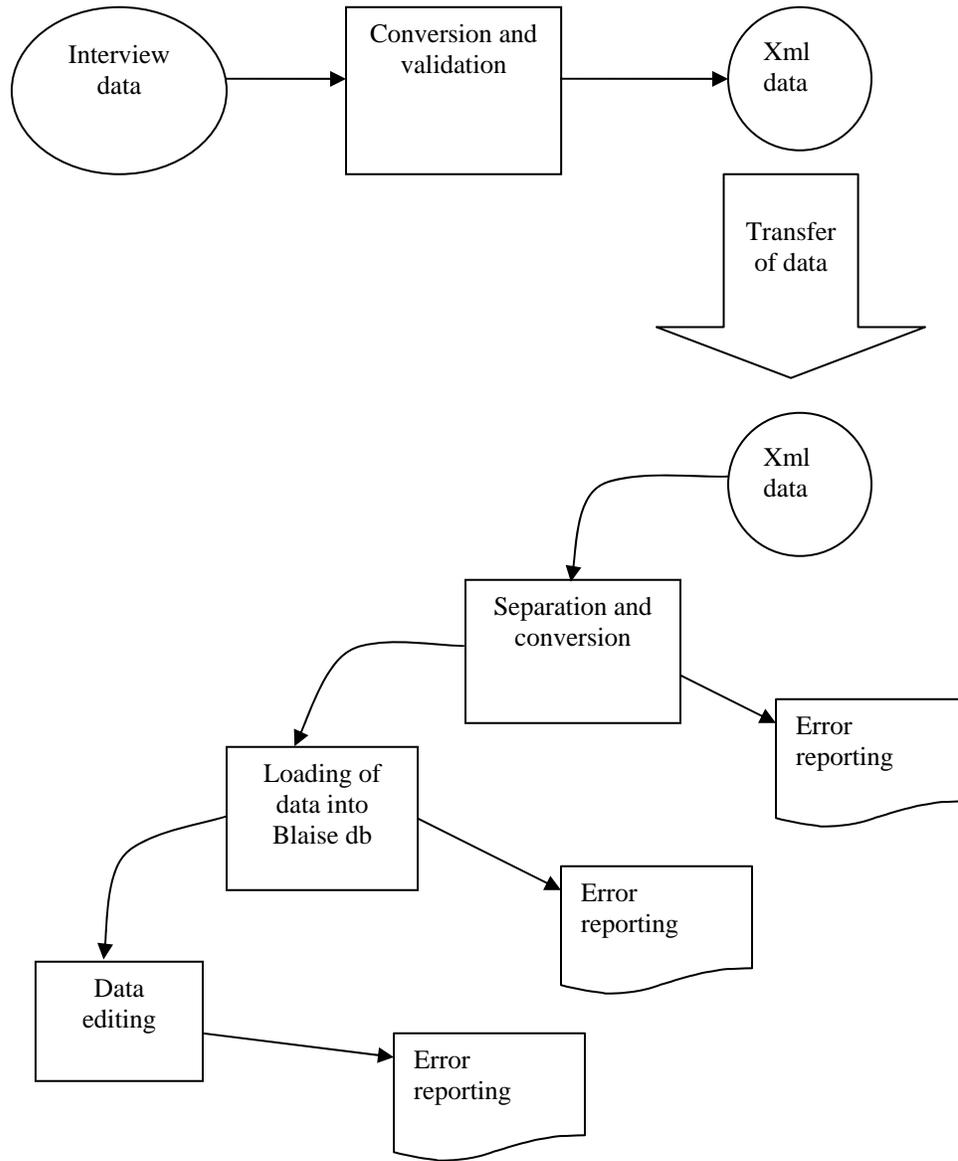


Fig. 1: Overall architecture of the data capturing system for the LFS

The Interview Consortium collects data from interviewers – from Telephone as well as Personal interviewing – on a daily basis. These data are converted into an xml document and validated towards the xml schema. The data are validated separately for each active survey, i.e. in the beginning of each quarter some data may belong to the stratum of the previous quarter. Also, a pilot study with interviews in yet another questionnaire may

possibly be part of the data stream. These data are combined into one single document ready for transfer (fig. 2).⁴

```
<Forms>
  <Form_Lfs2007q2> ... </Form_Lfs2007q2>
  <Form_Lfs2007q2> ... </Form_Lfs2007q2>
  ...
  <Form_Lfs2007q3> ... </Form_Lfs2007q3>
  <Form_Lfs2007q3> ... </Form_Lfs2007q3>
  ...
  <Form_MyPilotStudy> ... </Form_MyPilotStudy>
  ...
</Forms>
```

Fig. 2: Xml document

2.4 Processing

After receiving the xml document some processing is carried out:

1. It is split up into separate xml documents – one for each actual survey.
2. Some data in the xml document is converted into a format suitable for reading into a Blaise database.
3. The data are actually read into a Blaise database (Interview format).
4. The data are automatically coded and converted to another Blaise database (Editing format)
5. Dirty forms are edited (primarily coding of Occupation, Business and Education)

From each of these steps certain automatic or semiautomatic processing is carried out. The automation is heavily based on strict naming conventions for xml elements and file names.

2.4.1 Separation of xml document

On receipt of the xml document the first step is reading the document and separating into portions for each survey. This process is carried out by a java script that reads the data as a DOM document and writes the data into separate xml documents based on the

⁴ Eventually, the data were split into two documents – one for telephone interviews and another for personal interviews, due to the different handling of the two kinds of interviews by the Interview Consortium and different deadlines for completion of interview strata dependent on the interview mode. Telephone interviews must be completed by one and a half week after the period questioned, personal interviews may be completed until four weeks after. Either of the xml documents, however, may contain forms from more than one version of the questionnaire.

Form_XXX-elements. The resulting xml documents are all named as the input file with the name of the data model appended.

LfsData_2007-07-26.xml →

LfsData_2007-07-26_Lfs2007q2.xml + LfsData_2007-07-26_Lfs2007q3.xml + ...

From this process an error may be reported if the xml document is not well formed, or if the resulting documents cannot be validated towards the respective schemas.

2.4.2 Conversion of unsuitable values

Certain fields of the data model (stratum data) allow enumerated values with spaces. This is perfectly legal in xml and also according to the OIO xml definition.

As an example, in OIO xml the marital status is defined like this:⁵

```
<schema
  targetNamespace="http://rep.oio.dk/cpr.dk/xml/schemas/core/2005/09/19/"
  elementFormDefault="qualified" version="1.1">
<element name="MaritalStatusCode" type="cpr:MaritalStatusCodeType">
  <annotation>
    <documentation>
      Identifies the marital status and whether the person is alive or dead.
    </documentation>
  </annotation>
</element>
<simpleType name="MaritalStatusCodeType">
  <restriction base="string">
    <enumeration value="married"/>
    <enumeration value="divorced"/>
    <enumeration value="widow"/>
    <enumeration value="registered partnership"/>
    <enumeration value="abolition of registrered partnership"/>
    <enumeration value="longest living partner"/>
    <enumeration value="deceased"/>
    <enumeration value="unmarried"/>
  </restriction>
</simpleType>
</schema>
```

Fig. 3: OIO definition of Marital Status

An example data field therefore could comprise the following value:

```
<MaritalStatus>longest living partner</MaritalStatus>
```

However, these values are not well suited for reading directly into a Blaise enumerated field, so these field values are changed in advance. This is also carried out by the same java script as mentioned above.

⁵ From: The OIO Infostructurebase

As an impact of our efforts to conform to the OIO standard we implemented an XML language in the Blaise data model in order to represent references to the OIO.

Until now only two fields of the LFS data model are affected, so it's still manageable. It should be possible, though, to write a Cameleon script that could automatically generate a java script for carrying out this conversion based on the defined Blaise data model including XML language texts.

2.4.2 Conversion into Blaise

Conversion of the xml document into a Blaise database is carried out by a program written in C# and using the Blaise API. The program traverses the xml document and for each value the proper field is imputed, for example, the following data:

```
<MyBlock><MyField>MyValue</MyField></MyBlock>
```

should be assigned as:

```
MyDB.Field("MyBlock.MyField").Text = MyValue;
```

From this process error reporting comprises illegal field values as well as illegal (unknown) field names etc.

2.4.4 Automatic coding

The process of automatic coding is a well-established process in the LFS post processing in Statistics Denmark. The error reporting from this process mainly is about unsuccessful coding and may be used to refine the automatic coding process or to produce feed-back to interviewers about ambiguous descriptions etc.

2.4.5 Data editing

The data editing instrument is primarily developed for carrying out coding of Occupation, Business and Education.

However, this editing tool also showed to be the place systematic errors in routing or data conversion from the side of the Interview Consortium has been discovered.

3. Overall experience of the process

First of all, the project has been a success in the sense that we have managed to outsource interviewing for a large survey and still get data that are comparable to what we would have gained from continuing the interviewing in-house.

Obviously, there is a risk of losing the control of the process when outsourcing a core process like interviewing in the data capture process. Assessment of the quality of data is therefore a crucial part of such a project.

After six months of operation the daily exchange of data works all right, though there have been examples of misinterpretations detected – especially in connection with set up of new versions of questionnaire.

One of the most important challenges is that an xml schema may assure correct format of the data, but not the integrity (rules). Some routing errors have thus been detected when reading the xml document into the Blaise database for editing.

Considerations for improving the control of data therefore imply means for combining the validation against the xml schema with other validations.

Some research is now being carried out in this field.

4. Case study: The differences and similarities between Blaise 4.8 xml and Statistics Denmark exchange formats

Noticeable, the xml data files generated by Blaise 4.8 and the Statistics Denmark exchange format are very equal in structure – probably due to the fact that Blaise data are hierarchical by nature and therefore by same nature should be converted into xml while retaining this hierarchical structure.

The first difference that appears to the viewer is that Blaise 4.8 xml uses a document element <Dataset> containing one <Datarecord> element for each form in the dataset. The similar elements in the Statistics Denmark exchange format are <Forms> and <Form_DATAMODELNAME>, respectively.

The reason for naming the record element with the name of the data model is due to a requirement that one single xml document should be delivered from the supplier, possibly containing forms from (at least) two consecutive surveys (or pilot studies). Thus, this naming model allows the supplier to deliver one xml document with e.g. contents like this:

```
<Forms>
  <Form_Lfs2007q2> ... </Form_Lfs2007q2>
  <Form_Lfs2007q2> ... </Form_Lfs2007q2>
  ...
  <Form_Lfs2007q3> ... </Form_Lfs2007q3>
  <Form_Lfs2007q3> ... </Form_Lfs2007q3>
  ...
</Forms>
```

When receiving this document it's quite easy to split it into a number of xml documents each containing forms of only one data model for further processing.

When looking at the interior of the form/datarecord the elements are named in both formats by the field or block instances:

```
<MyBlockInstance>
  <MyFirstField> ... </MyFirstField>
  <MySecondField> ... </MySecondField>
  ...
</MyBlockInstance>
```

The actual values, however, are supplied slightly differently. In the Blaise 4.8 xml format the values are supplied as simple text between the name element tags, for example:

```
<Age>48</Age>
<Gender>Male</Gender>
<CivilStatus>Married</CivilStatus>
```

While the Statistics Denmark exchange format would supply these values like this:

```
<Age><value>48</value></Age>
<Gender><value>Male</value></Gender>
<MaritalStatus><value>Married</value></MaritalStatus>
```

Why? Well, look at how non response values are supplied. In the Blaise 4.8 format non response values are supplied through element attributes.

The Statistics Denmark exchange format, however, conforms to the recommendation from the OIO committee that element attributes should be avoided whenever possible.

```
<Age><nonResponse>EMPTY</nonResponse></Age>
<Gender><nonResponse>DONTKNOW</nonResponse></Gender>
<MaritalStatus><nonResponse>REFUSAL</nonResponse></MaritalStatus>
```

Set values are rather equally filled:

```
<MySetField blaiseContainerType="set">
  <choice>blue</choice>
  <choice>green</choice>
  <choice>yellow</choice>
</MySetField>
```

compared to:

```
<MySetField>
  <value>blue</value>
  <value>green</value>
  <value>yellow</value>
</MySetField>
```

Notice, that when reading these data into Blaise the information that the field is of type set is not needed. Why? Because, when reading data into a Blaise database the Blaise datamodel is available and can tell the xml reader that this field is a set field.

Similarly, non response values are supplied just like other kinds of fields, respectively:

```
<MySetField xsi:nil="nil" blaiseNonResponseType="DONTKNOW" />
```

```
<MySetField>  
<nonResponse>DONTKNOW</nonResponse>  
</MySetField>
```

Array fields have different layouts in the two formats:

```
<MyArrayField blaiseContainerType="array">  
<MyArrayFieldItem blaiseIndex="1">  
    myvalue1  
</MyArrayFieldItem>  
<MyArrayFieldItem blaiseIndex="2">  
    myvalue2  
</MyArrayFieldItem>  
    ...  
</MyArrayField>
```

Compared to the somewhat simpler:

```
<MyArrayField><value>myvalue1</value></MyArrayField>  
<MyArrayField><value>myvalue2</value></MyArrayField>
```

Just like in the case of set types when reading data into a Blaise database you know from the data model that this field is an array and the proper index may be supplied automatically.

You may still supply empty array field, e.g.:

```
<MyArrayField><value>myvalue1</value></MyArrayField>  
<MyArrayField/>  
<MyArrayField><value>myvalue3</value></MyArrayField>
```

Or:

```
<MyArrayField><value>myvalue1</value></MyArrayField>  
<MyArrayField><nonResponse>EMPTY</nonResponse></MyArrayField>  
<MyArrayField><value>myvalue3</value></MyArrayField>
```

References

OIO website, <http://www.oio.dk> [last viewed July 27, 2007]

OIO Infostructurebase, <http://isb.oio.dk/info> [last viewed July 27, 2007]

Michigan Questionnaire Documentation System (MQDS): A User's Perspective

Heidi Guyer and Gina-Qian Cheung, The University of Michigan

1. Background and Introduction

In 2003, Survey Research Operations (SRO), a unit within The University of Michigan's Institute for Social Research Survey Research Center, initiated the development of a system that would allow users to generate questionnaire documentation from a Blaise data model and codebooks from a Blaise data set. Prior to the development of the tool, it was difficult, if not impossible, to generate questionnaire documentation from a Blaise data model. In most cases, users would first design and specify a questionnaire in a word processing program, such as Microsoft Word. Blaise programmers would then generate a data model based on the information provided in the document. Further changes such as changes to question wording, code frames or logic had to be made both in the documentation as well as in Blaise. Documenting changes that had occurred in a data model once it was fielded was incredibly cumbersome and a feat requiring patience and skill. One of the primary goals of the system developed by The University of Michigan was to allow users to generate the documentation based on what was actually programmed (not just what we hoped or planned to have programmed), thus providing sound survey documentation. Such documentation can be used for testing, training, questionnaire development, and study documentation purposes. Likewise, generating a codebook from a Blaise data set was also often difficult and time-consuming and usually involved more than one application to produce the codebook.

Between 2005 and 2006, MQDS was re-written in .NET and additional utilities were added. These included XML merge and the re-apply stylesheet features. A thorough MQDS User Guide was developed and is accessible from within the system. In 2006, a consortium of international users and affiliates was formed to provide support for further development and enhancements of the system. The international consortium included Statistics Canada, Mathematica Policy Research, Inc., the National Centre for Social Research in the UK, and The University of Michigan's Survey Research Center.

In July 2007, MQDS Version 2.5.0.0 was released to the consortium of users. This latest release concluded the development phase of the Michigan Questionnaire Documentation System.

2. Methods

We attempted to evaluate the main components of the system as well as the new tools available within the system by asking users to test the various utilities using a wide range of data models. Usability testing has been described as measuring how well test subjects respond in four areas: time, accuracy, recall, and emotional response⁽¹⁾. While we were not conducting usability testing per se, the concepts apply as our goal was to assess MQDS from the standpoint of usability, via systematic user feedback from testing complex data models. The tools and features evaluated included Questionnaire Documentation, Codebook from Blaise, Codebook from SAS, XML Merge, Reapply Stylesheet, Blaise to SAS, and SAS to XML.

Software testing is described in Wikipedia as, "... a process of technical investigation, performed on behalf of stakeholders, that is intended to reveal quality-related information about the product with respect to the context in which it is intended to operate. This includes, but is not limited to, the process of executing a program or application with the intent of finding errors." ⁽²⁾ With this in mind, a series of studies were selected to test the latest MQDS release. The studies were selected based on several criteria, such as size and complexity of the data model, languages employed and the need for using external databases to process complex fills, with the goal of providing a range of data models. This would allow us to determine when MQDS is at its optimal processing versus when problems may begin to occur.

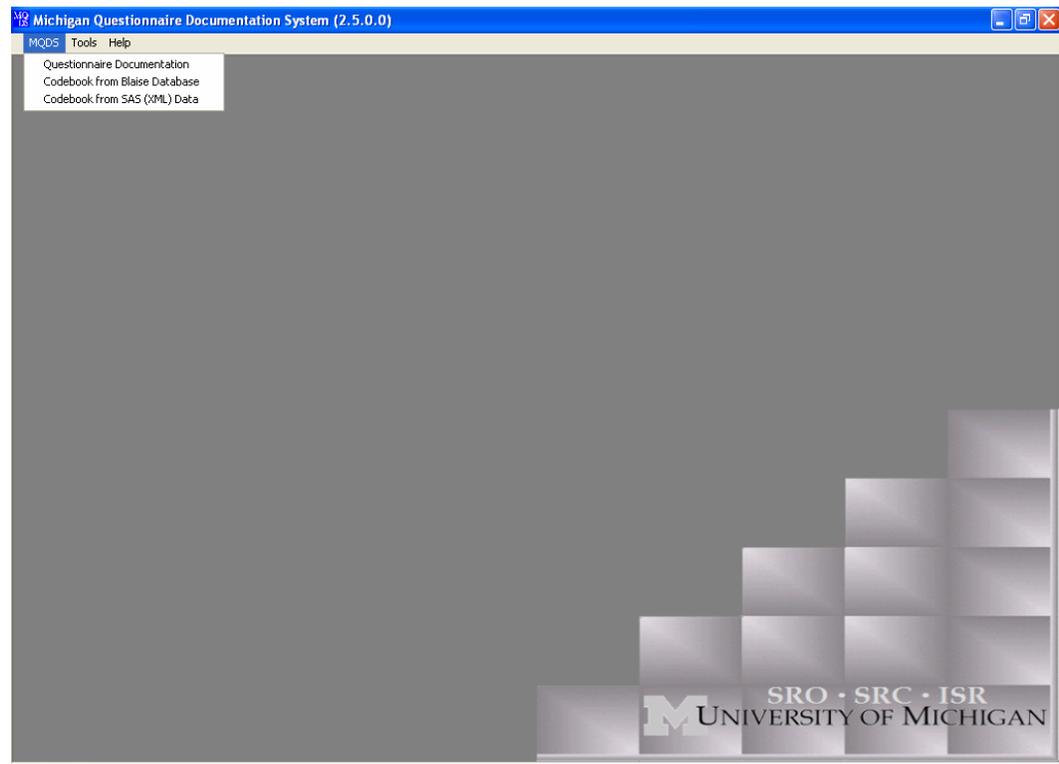
The three primary components, Questionnaire Documentation, Codebook from Blaise and Codebook from SAS (XML) were evaluated using data models of varying complexity, in multiple languages and with a range of case sizes. Each of these components allows the user to produce output in HTML as well as in RTF format. Both of these features were evaluated as well. The four new tools (XML Merge, Reapply Stylesheet, Blaise to SAS and SAS to XML) were tested and evaluated from a usability perspective as well.

In this paper, we will focus on providing an overview of the features of the system and provide a critical look at the various components and usability from a user's perspective. We comment on the interface, the instructions provided (i.e. help files), whether the expected output was generated, as well as the overall usability of the tool. The comments included in this review are limited to those related to basic functionality from the user's perspective. While the comments are not technical in nature, the authors' desire is to express the importance of developing straightforward, user-friendly software. Oftentimes, this can only be accomplished when clearly defined system specifications are developed before undergoing the system design, a multi-disciplinary team is involved in the specifying and testing of the system, and rigorous testing methods are employed.

3. MQDS Interface Review

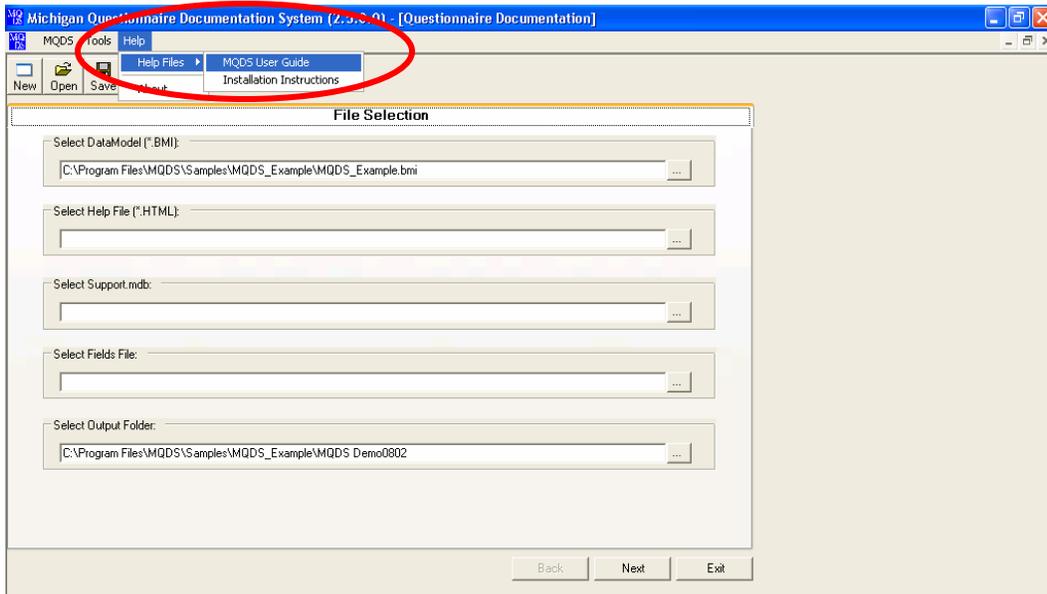
The first utility within MQDS reviewed was the Questionnaire Documentation tool. The utility is relatively straightforward and easy to use although it was necessary to consult the User's Guide several times in order to produce the desired output.

Figure 1. Accessing the Questionnaire Documentation Tool



The first example provided in this review is the questionnaire output generated using the MQDS example project which is available to all MQDS users as part of the installation package. It is strongly encouraged that first time users review the tools and requirements with the User Guide in hand and by using the MQDS Sample project to gain familiarity with the program. The data model associated with the MQDS example project is small enough that output can be generated quite quickly and the user can then determine the features to include in their own output. The User Guide can be accessed via the Help drop down menu as shown in Figure 2.

Figure 2. Accessing the MQDS User Guide



When generating Questionnaire Documentation, the first screen that you arrive at is the file selection screen (Figure 3). Although there are five different files that can be selected--Data model, Help File, Support.mdb, Fields file and Output folder--there are only two files that are mandatory to run the documentation: the Blaise data model and the output folder. Two others add increased value to your output, the Help File and the Support.mdb, but are not necessary. The Fields file is only used if you've generated questionnaire documentation in MQDS in the past and would like to speed the process up. The information regarding the mandatory versus optional files, and a description of each of the files, can be found in the User Guide. It would be useful to have an indication on the screen as to what is required versus what is optional.

When selecting an output folder, you are taken to the folder where your data model is stored (Figure 4). The option to make a new folder is presented as well. Both of these features are useful as there are so many steps along the way, it is easy to lose track of where your output is actually stored by the time it is produced.

Figure 3. File Selection Screen

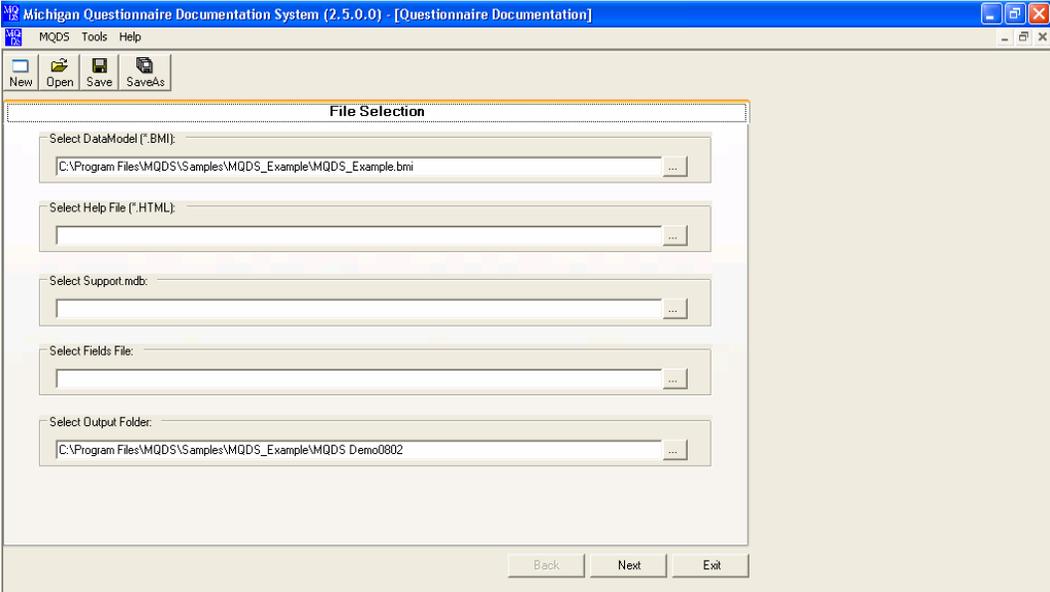
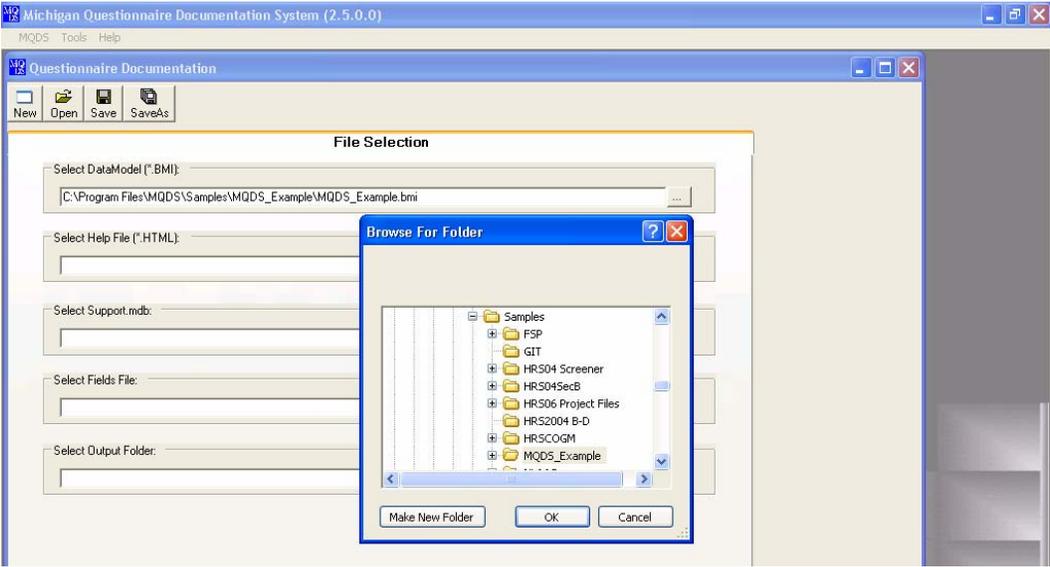


Figure 4. Output Folder Selection Screen

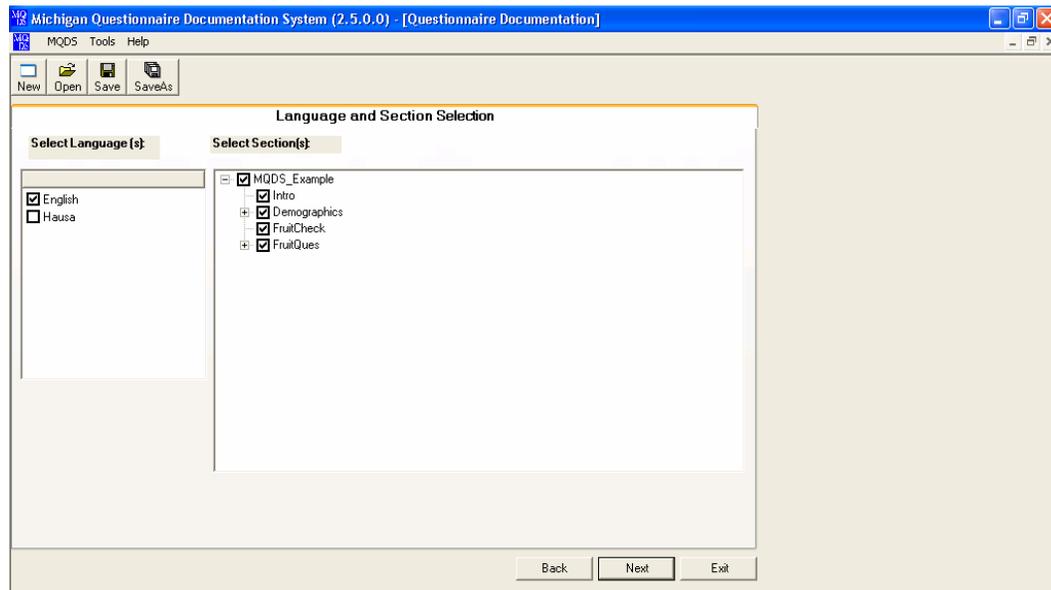


The Language and Section Selection screen displays all of the languages in which the Blaise instrument is programmed in, as well as all of the instrument sections (Figure 5). A tree feature has recently been added to the section check list which allows the user to select certain questions within a section. This increased user functionality allows the user a wide range of options- from generating documentation for a full instrument to generating output for a single question. Although the latter situation will not often present itself, it can be useful when documenting changes to particular questions across data models or across waves of a longitudinal study.

It should be noted here that although the user can limit the output to the selection of certain sections, the system will continue to process the entire instrument but only the selected sections will actually be displayed in the output. Limiting the number of sections you are outputting in any given run is a useful technique when processing questionnaire documentation for large or complex data models. Depending on the size and complexity of the data model, this may be the only option for producing output from the Michigan Questionnaire Documentation System.

Thus far, it has been possible to produce output in up to five languages for one survey instrument. The National Latino and Asian American Study (NLAAS) was programmed and conducted in English, Spanish, Vietnamese, Chinese (two dialects) and Tagalog (language of the Philippines). Depending on the language and the fonts currently installed, it may first be necessary for the user to install language-specific fonts on the computer used to generate MQDS output. If possible, the same font used when programming Blaise should be installed. This will ensure that the text is accurately displayed as it appeared in Blaise in the selected language. Users should be aware that this is particularly true for Asian fonts such as Vietnamese or Chinese.

Figure 5. Language and Section Selection



The Stylesheet Selection screen is a crucial screen in determining the content and format of the output you are generating (Figure 6). It is here that the user will determine the elements included in the output, as well as (unknowingly) the amount of time it will take to generate the output. This last part is actually an unknown as it is dependent on many factors including the number of variables in the data model, the complexity of the logic, the number of languages you are attempting to produce output for at one time, where the fills will originate from and whether you are attempting to process a data model with logic (gotos) or without. Runtimes can vary from less than five minutes for a simple instrument with basic or no logic to over 24 hours for long data models with complex logic and fills. For example, the Health and Retirement Study (HRS) data model is 10,246 kb in size and must be run in sections at a time as it is too large to generate output for the entire data model at once. Even with running blocks of sections separately and without including logic, it takes close to 90 hours to generate output for the entire instrument.

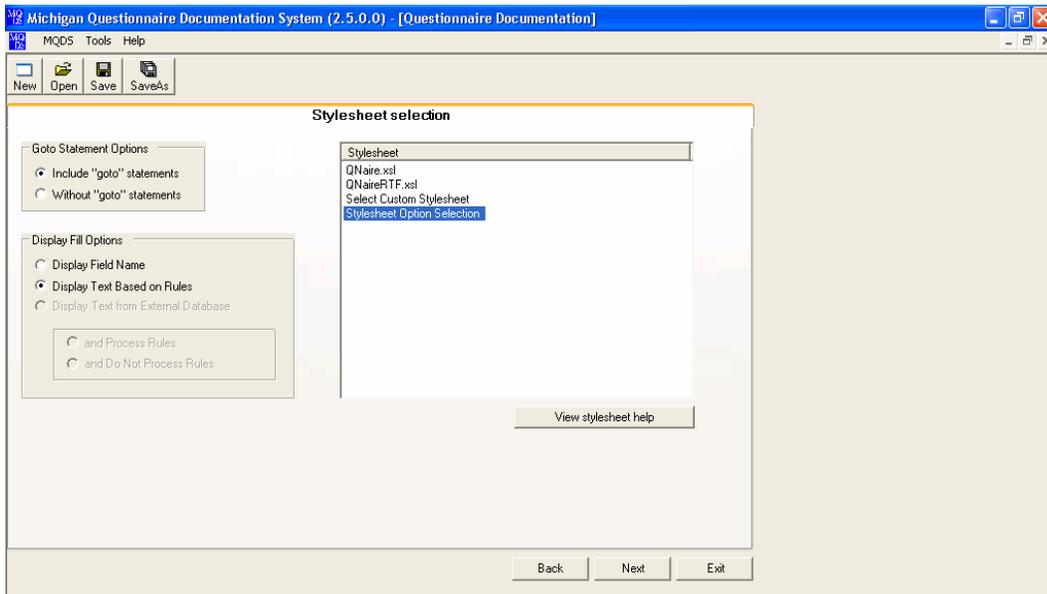
The menu selection titled “Display Fill Options” offers the user several ways of displaying fill text. For example, the user may want to display the field name (V232_Relationship) or the text itself based on the rules specified in Blaise. In the latter case, all fill text options will be displayed dependent on how this is actually specified in the Blaise data model. For example, a fill that shows the relationship to the respondent may be displayed as (Spouse / Partner / Sister / Brother / Mother / Father / Friend / Neighbor / Cousin / Other Relative / Other). This may be displayed for English alone, or in all languages in which the instrument is programmed. Another option is to develop a support database in Access which can be linked to the MQDS system when the fills are processed. If the logic behind the fills is particularly complex or the data model is particularly large or complex, this may be the route to take as the system will not process the fills (thus it will not drain the memory or diminish computer processing speed). Instead it will simply search for the variable itself in the support table and display the text exactly as it appears in the support table. While this is a convenient option for large and complex data models, generating the table itself may not be as easy as it sounds.

A very important, although somewhat confusing, feature of this selection screen is the option of selecting the type of stylesheet that will be generated. There are four options for users:

- Generating a default questionnaire in html format (QNaire.xml)
- Generating a default questionnaire in RTF (QNaireRTF.xml)
- Generating a questionnaire using a stylesheet that the user previously defined (Select Custom Stylesheet) or;
- Customizing a new stylesheet (Stylesheet Option Selection)

Those that are savvy in XML programming may be able to generate a custom stylesheet in which the items are displayed according to the user’s preference (Select Custom Stylesheet). Otherwise, there are three (3) options for questionnaire and codebook documentation. There is an advantage to using RTF output in that it can be edited and modified in a text editor such as Microsoft Word. The primary advantage of HTML output is that the format of the output itself appears exactly as it appears to the interviewer on the Blaise screen and hyperlinks are fully supported.

Figure 6. Stylesheet Selection Screen



When customizing a new stylesheet (Stylesheet Option Selection), the system allows the user flexibility in selecting the output to be displayed. For example, certain types of questions can be displayed, the universe can be displayed on the screen or embedded as a hyperlink and options for displaying logic are available as well. This is a relatively recent development of the system and will benefit from additional testing. It is possible that there are other widespread needs that could be incorporated in the stylesheet option selection list. Users can also control the display of links (Figure 8). Options include selecting the text color of the links as well as whether links are underlined or not. This feature is useful given that most interviewer instructions are programmed in the same shade of blue as hyperlinks. Thus, in order to distinguish between actual hyperlinks and displayed text, one may wish to change the color of the links (although this is not as intuitive when viewing the output) or underline the links.

Figure 7. Stylesheet Options Screen

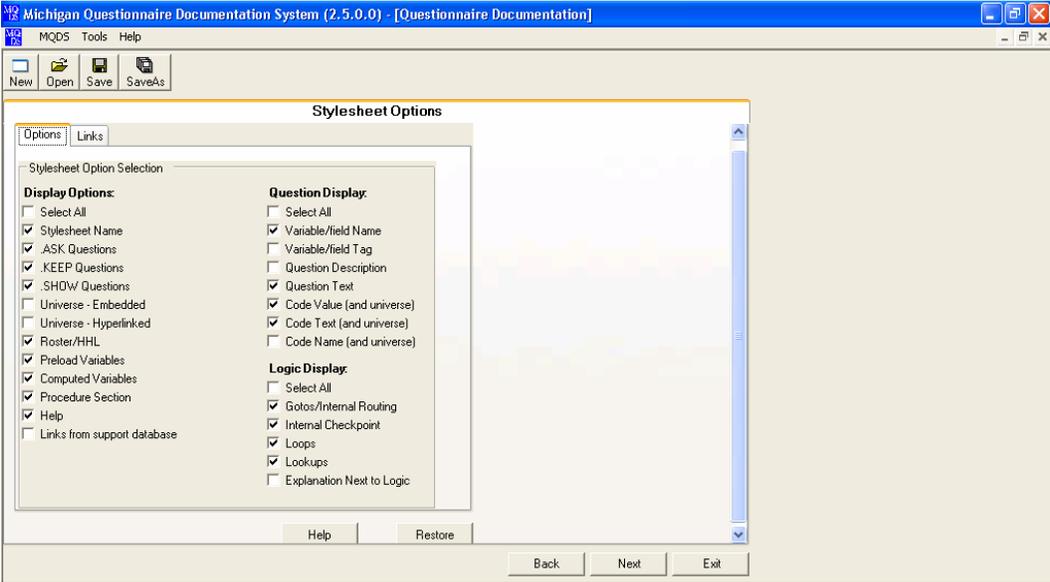
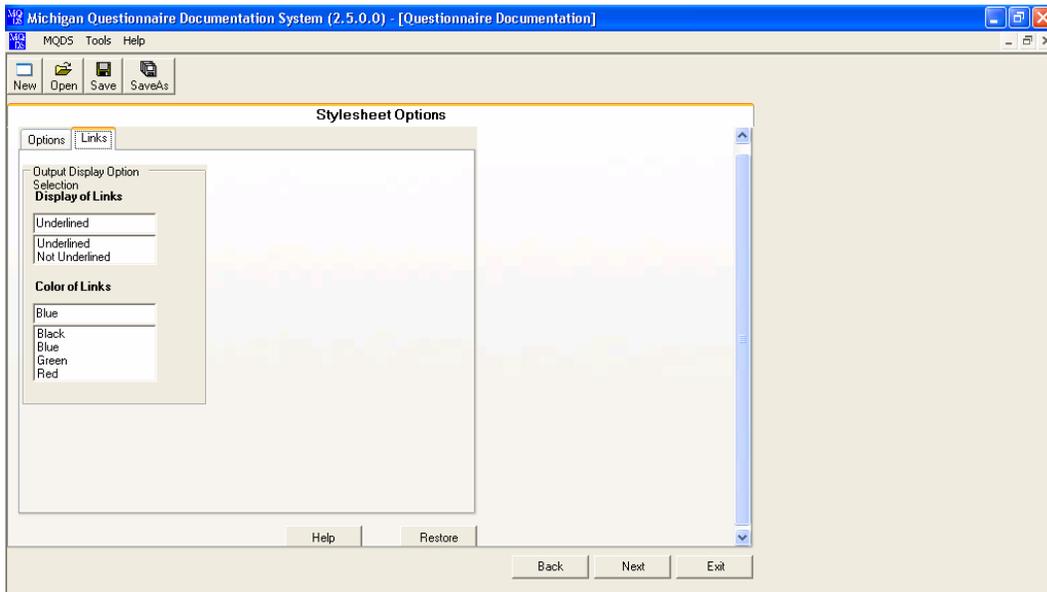
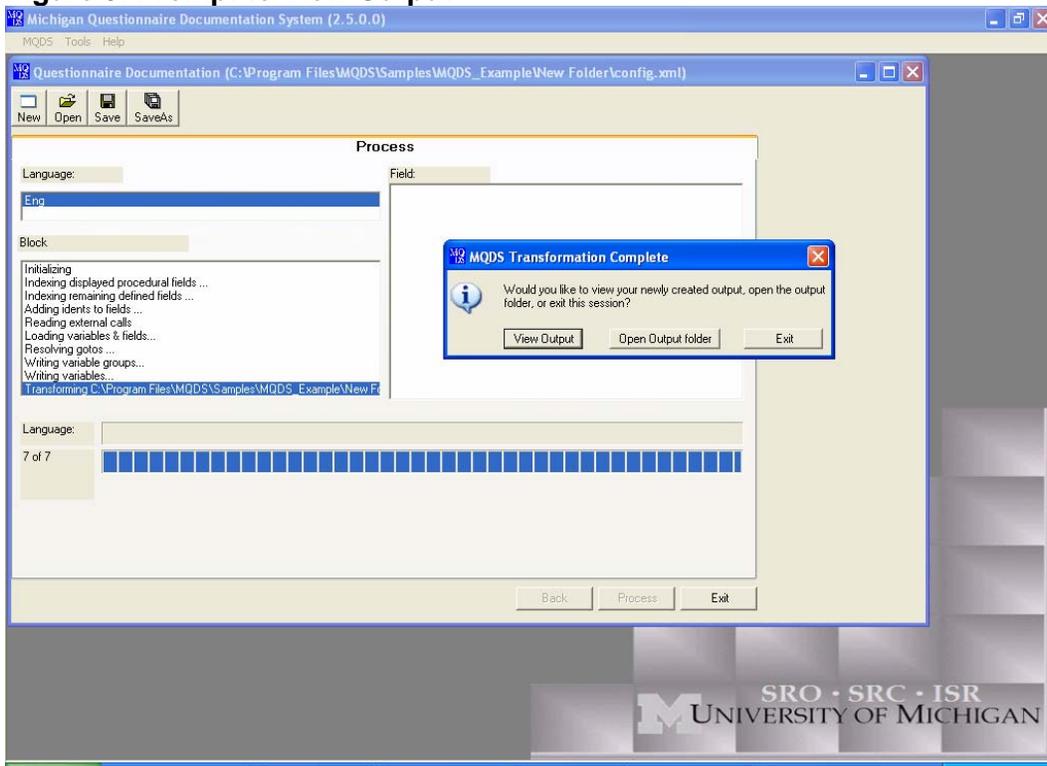


Figure 8. Stylesheet Options Screen 2- Links



Another recent feature of the system is a prompt to the user once processing has completed allowing the interviewer to either view the output, open the output folder, or exit the system (Figure 9). In the past, difficulties in locating the output were sometimes encountered, especially after particularly long processing times. This new prompt takes the user directly to the output.

Figure 9. Prompt to View Output



4. Comments on MQDS Output

4.1 Questionnaire Documentation

The HTML MQDS questionnaire output is easy to read and navigate, as well as being appealing to the eye (Figure 9.). The primary pane displays the question text, interviewer instructions, code frames and other links exactly as it does in Blaise while also displaying the sample universe, gotos, and unasked questions (.SHOW, .KEEP) as specified. Following the SRC's Blaise screen standards, the background is a cream color, question and response text are in black, and interviewer instructions are displayed in blue. You can easily jump from one section to the next using the links displayed sequentially, and by language, in a column on the left-hand side. Links to the User Guide are also easily identified. The ability to produce questionnaire documentation in non-English languages has a myriad of uses including the ability to compare question and response text programmed in English to the translation of the English text. (Figure 10)

Figure 10. MQDS Questionnaire Output

Michigan Questionnaire Documentation System (C:\Program Files\MQDS\support\Qnaire.xml)

Sample Questionnaire for MQDS

User Guide

Eng
DEMOGRAPHICS
FRUITQUES

Hau
DEMOGRAPHICS
FRUITQUES

In which town do you live?

CHILDREN_IC_1
(Gender = 2 Female) AND (Age > 15)

1 EXPR IS FALSE GOTO FruitCheck

2 EXPR IS TRUE

Children

How many children have you given birth to?

0 - 25

Universe

FruitCheck

Do you like to eat fruit?

1 Yes

5 No GOTO_End

Figure 11. MQDS Questionnaire Output in Hausa (non-English language)

Michigan Questionnaire Documentation System (C:\Program Files\MQDS\support\Qnaire.xml)

Sample Questionnaire for MQDS

User Guide

Eng
DEMOGRAPHICS
FRUITQUES

Hau
DEMOGRAPHICS
FRUITQUES

Wannan gari ne?

CHILDREN_IC_1
(Gender = 2 Matace) AND (Age > 15)

1 EXPR IS FALSE GOTO FruitCheck

2 EXPR IS TRUE

Children

Yara nawa ka ayhwa?

0 - 25

Universe

FruitCheck

Kana so ci kayan garka?

1 Yi

5 No GOTO_End

While the default stylesheet includes most options you would typically want to display in questionnaire documentation, the ability to customize a stylesheet provides the user with flexibility in determining the features to display. The user is able to customize the output taking into account what the documentation will be used for as well as how the instrument was programmed. For example, if you don't want to display the universe but want it embedded and accessible via a hyperlink, -or not display it at all if you are only interested in the question text itself. Likewise, the user may choose to display the variable name or the variable tag, or both, depending on how informative each may be.

While the hyperlinks to such things as the sample universe, question level help text, and gotos are incredibly helpful, the fact that interviewer instructions are displayed in the same blue font as the hyperlinks can lead to confusion. Regardless, the ability to insert hyperlinks for a wide range of information has huge potential. For example, it could be possible to provide links to screen shots or sound and video files..

Another feature of the output that is incredibly useful is the display and hyperlink to the “gotos”. However, users should note that selecting this option increases the processing time exponentially. In fact, a data model can literally take hours to process if the logic is complex.

One drawback to the system is that it can be difficult to determine the accuracy of the output. For example, determining whether the gotos are correct (do they take you to the right place in the instrument), the universe is correctly displayed, and whether all of the questions are actually displayed. Reviewing and comparing the output can be a cumbersome, time consuming process. It would be useful to develop a tool allowing one to compare the variables in the output with the variables in the data model itself to ensure that the output is complete, or that all the variables one would expect to see are displayed.

4.2 Codebook (from Blaise and from SAS)

As with the questionnaire documentation, the codebook output is also displayed as it is on the Blaise data entry screen. The user is also offered the same flexibility as with the questionnaire documentation in determining the elements to include in the output and how they should be displayed (hyperlinks or embedded, the color of the hyperlinks, etc.). In addition to the questions and codeframes, frequencies are displayed for categorical variables and basic descriptive statistics (mean, median, standard deviation) are displayed for continuous variables. The display of open-ended responses is not as useful thus it's preferential to recode this type of response into categorical variables.

As with the questionnaire documentation, codebooks can be generated in all languages in which the instrument is programmed. This feature may be especially useful if separate SAS data sets are developed by language of interview. Codebooks could then be generated for the selected study population in the actual language of the interview.

Just as with the questionnaire documentation, it would be helpful to have a utility that would compare the variables in the output to those in the source data set to ensure that all are included, none have been excluded, and that there are no duplicates.

When generating a codebook from a SAS data set using a recoded data set, users should be aware that there may be discrepancies between the MQDS output and newly constructed and re-coded variables. This is because the question text and response categories that are displayed are derived from the Blaise data model. Thus, the question text for newly constructed variables will not be available for display. Likewise, there will be a discrepancy between the response categories displayed and the response categories for which frequencies are provided when variables have been re-coded or response categories collapsed. In this situation, it is advisable to generate RTF output, rather than HTML output, to edit the output in a word processor after it's processed. Descriptors and clarifications can then be added to the codebook output.

4.3 Other utilities: XML Merge, Reapply Stylesheet, Blaise to SAS and SAS to XML

The XML Merge and Reapply Stylesheet utilities both offer the user a way to optimize multiple runs of the same instrument. XML Merge provides a means to combine the outputs from multiple "runs" of MQDS when a data model is too large or complex to be processed in its entirety at one time. The Reapply Stylesheets utility allows the user to choose different output, such as RTF when HTML was initially chosen, or to produce a questionnaire when a codebook had been run. Customization of output is also available in the same way as the MQDS wizard. The benefit is MQDS does not need to be run a second time for different output. Thus, these are time saving tools- which are always appreciated.

The Blaise to SAS and SAS to XML utilities allow the user to generate a SAS data set from the Blaise data. The SAS data set can then be used for data analysis or to generate a codebook in MQDS. To generate the codebook, one additional step must be taken which is to run the SAS to XML utility. As with the other utilities, it would be useful to have a tool that ensures all of the variables in the Blaise data model are in fact included in the SAS data set.

5. Summary

In summary, the Michigan Questionnaire Documentation System (MQDS) is a useful tool that generates quality output. The system provides the user with flexibility in determining the content and appearance of the output, both for Blaise questionnaires as well as for codebooks. The fact that the output generated by the system replicates the actual appearance of the questions on the Blaise screen encourages the use of the system by users with non-programming or technical backgrounds, as well as programmers. For example, MQDS output can be used during the questionnaire development phase to generate questionnaire documentation for instrument testing. Reviewing the documentation allows users to identify errors ranging from misspelled words, incorrect interviewer instructions, incorrect fill text, problems with data models programmed in multiple languages (i.e. English text displayed in a Spanish instrument), miscoded response categories, to problems with the flow of a questionnaire.

The primary drawbacks to the system are centered on larger or complex data models or data sets. These drawbacks include:

- The amount of memory necessary to run larger data models: at times we've had one personal computer dedicated solely to running selected sections of a data model overnight.
- The processing speed for larger or complex data models. Due to the way in which MQDS uses memory, the processing speed can be very lengthy. In fact, at times the system has stopped with no indication that it has not been able to fully process the data model. Because the user has become accustomed to the lengthy processing time, you may be unaware that MQDS has stopped processing.
- The inability to process complex logic on larger or more complex data models. This aspect of the system has been somewhat unstable. Previous versions of the system seemed to process logic (gotos) more efficiently whereas it is no longer possible to generate output with the logic displayed for larger data models.

While no further development is currently being undertaken on MQDS, there are several areas that may be explored further. This includes enhancing a comparison grid developed to allow users to compare multiple data models. The system has been used to compare data models across three major mental health studies from the Collaborative Psychiatric Epidemiology Studies (CPES), as well as codebooks with a crosswalk for the three studies. Additionally, the output was generated in all languages in which the instruments were programmed and comparisons can be made across languages as well. This feature is useful for testing purposes, when comparing actual question text on studies with similar questions, as well as for cross-wave comparisons on panel studies.

Additional areas of future development should include the incorporation of the forthcoming Data Dissemination Initiative (DDI) Version 3 recommendations. The DDI is an XML specification for social science metadata that is being developed by an international group called the DDI Alliance. A release of DDI version 3.0 is planned for this fall. This would allow for DDI 3 compliant survey documentation. The possibility of merging MQDS with Blaise's Delta has also been explored. Doing so would allow MQDS to display enhanced tree views, a detailed view plus a graph view, of the BMI.

In this paper, the primary usability features of MQDS have been outlined as well as areas for future consideration. Rigorous testing of the system was recently undertaken and the documentation has been much improved. In conclusion, the system is relatively robust and the recent developments are much appreciated by MQDS users.

6. Bibliography

1. Wikipedia. Usability Testing
2. Wikipedia. Software Testing
3. Michigan Questionnaire Documentation System: User's Guide. The University of Michigan. 2007.

The use of mobile devices to carry out Blaise surveys at the Office for National Statistics

Tim Burrell, Office for National Statistics, UK

1. Introduction – Why start to investigate now

ONS uses Blaise for all its social survey data collection. At present, interviewers on all surveys are equipped with a laptop to enter data. It has been a goal for ONS to take advantage of hand-held technologies for some time now for both household surveys and the International Passenger Survey (IPS). On the IPS data are collected via paper and input into Blaise via laptop (CADE). IPS interviews are carried out at ports, airports and international rail stations where data are very often collected on the move. For household surveys this must be taken into a household, set up and switched on before interviewing can take place. The use of hand-helds was first investigated over ten years ago but ONS felt that the hardware was not advanced enough to use. The prospect of Blaise 4.8 and the BASIL component, and the development of smaller, lighter, more powerful and more suitable hardware have made ONS look again at alternatives to laptops. In addition to improvements to the software and hardware available, there are also increasing demands on the types of data collected and the speed required from data collection to dissemination, which has contributed to this review.

2. Use of mobile devices in other organisations

2.1 Public sector

ONS produce many statistics for the United Kingdom (UK), requiring data from England, Scotland, Wales and Northern Ireland. When a survey requires Northern Ireland data we work in conjunction with the Northern Ireland Social Research Agency (NISRA). When working with NISRA the Blaise questionnaire is developed by ONS and sent to NISRA. They then make any necessary changes to the questionnaire to fit with their systems.

NISRA stopped using laptops four years ago and started to use tablet PCs. The first tablet used was a Fujitsu Stylistic which is no longer in production. NISRA currently use an HP TC1100. Once again, this device is no longer in production. NISRA is currently looking into replacement hardware.

Tablet PCs are preferred by NISRA as they consider them to be easier for interviewers, they are lighter than laptops and easier to carry. They also reported that some of the Blaise modelib features are very useful when using tablet PCs. e.g. the optional calculator screen

When NISRA moved its field force to use tablet PCs as opposed to laptops all interviewers had a two day training course. For new interviewers they found that training new interviewers in the use of tablet PCs is easier than laptops. Since introducing tablet PCs at NISRA, they have experienced less breakages.

2.2 Private sector

Mobile devices are being used for survey data collection in both the public and private sectors. Nowadays, in the UK, large, typically market research companies are doing more social surveys. This means they are carrying out longer, more complex questionnaires. Many of these research companies are using mobile devices. At IPSOS-MORI over 50 percent of their UK revenue is now gained from social research as opposed to market research.

The MORI part of IPSOS-MORI is currently using a Motion Computing tablet PC (some older Fujitsu tablet PCs remain in the field). The IPSOS part is using a Dell Laptop.

3. Prospective use of handhelds within ONS

3.1 The UK Census

The next UK Census is in 2011. The census form is a very short, simple form. Census Division within ONS are investigating whether the Census can be carried out using a mobile device. In addition to being used as a data collection device, Census enumerators could possibly use the hand-held for up-to-date case management of returned Census forms.

3.2 Door-step introductions and short surveys

Using a hand-held device, interviewers could perform easy and quick door-step surveys. These surveys could collect basic household information or could be used to carry out doorstep sifts to identify which adult is required for interviewing (on an individual household). This method would relinquish the need for interviewers to enter the respondent's home.

There are separate issues with carrying out doorstep surveys – a practice which is not encouraged at present due to security issues and confidentiality. It is not thought safe for interviewers to openly carry hardware in the street. It is also against confidentiality rules to ask some questions at the doorstep as opposed to in the respondent's home.

3.3 English Housing Conditions Survey (EHCS)

The EHCS comprises 2 parts. The first part is the household interview. Following this a surveyor visits to collect additional information on the condition and structure of the house. Currently this information is collected on paper. The data collection process involves the surveyor moving from one room to the next so using a laptop is not suitable. The data are then transferred from paper to Blaise. With a tablet PC or hand-held device the data could be input straight into Blaise.

3.4 International Passenger Survey (IPS)

The IPS carried out at ports, airports and international rail terminals to record people travelling in and out of the UK. Due to the high volumes of people travelling through these terminals, interviews are carried out on paper while interviewers are on the move and then transferred to Blaise. If hand-held technology were introduced time could be saved if the data entry process is removed. It would also lead to less people required on a

shift which will save money. At present 1 shift involves 3 interviewers interviewing and one coding (4 in total). If data collection input took place at the time of interview there would no longer be a requirement for an additional coder.

IPS interviewers use a room based at an airport in which to record the answers to a laptop. The survey is coming under increased pressure to give up these areas so they can be used as retail space by the airport. The Eurotunnel terminal currently at London Waterloo is moving to a newly built terminal at a different location in London where there has been no provision made for IPS interviewers

3.5 For all social survey data collection

In the future, as has been shown at other organisations, the whole of the ONS's social survey data collection could be carried out via a mobile device. One argument is that with the latest or more up-to-date technologies, interviewers look good, feel more professional and are seen as such if they are using the latest technology. Interviewers also feel more confident that they can 'sell' a survey.

4. Hardware options

There are three main types of mobile device which were identified. These are (1) hand-held, or PDAs; (2) a hybrid laptop and (3) a Tablet PC.

4.1 Hand-held or Personal Digital Assistants (PDA)

PDAs are typically hand-held instruments used for basic PC tasks such as checking emails, keeping track of calendars and can facilitate the use of some Microsoft functions such as MS Word and MS Excel. Compared to a PC, PDAs have very small screens. They are built to be lightweight and to fit in an individual's hand. It was thought that these small sized hand-helds (up to 8cm x 28cm) would be too small for ONS interviewers. There are also problems posed by questions either with a lot of text (or help text) or with a number of answer categories as they would not fit on the screen and scrolling on the screen would be required. This is against current ONS screen standards although other standards would require development for a smaller screened device. Options sought would include a different set-up of the questionnaire including less on-screen options and only one question per screen.

In addition to the small screen sizes, PDAs have minimal memory (32 – 128 MB) which would not be suitable for ONS surveys with large help or external files. PDAs are useful for some tasks, such as case management for census enumerators, or for very simple questionnaires and doorstep sifts, but it is not seen as a solution for all survey types.

4.2 Hybrid Laptops

A hybrid laptop can be used either as a traditional laptop or the screen, attached to the keyboard by a hinge, can be rotated so the device can be used as a tablet. Hybrids are built with the same screen size as a traditional laptop (approx 285 x 244mm).

Due to the presence of a keyboard which can not be detached, hybrids tend to be too heavy to be used for long periods of time when standing up (approx 5kg). Although hybrid laptops are available with accessories, such as shoulder straps, to aid the

interviewer when the device is being used in its tablet form, due to the weight of the keyboard element of the device, the extra weight does not allow for long-term continual use in the field, such as a two hour IPS shift. There are worries over how strong the hinge which joins the keyboard and screen are, which could lead to field breakages.

4.3 Tablet PC

Tablet PCs are available as lightweight devices (compared to laptops and hybrid laptops) at 2.5 – 5kg depending on screen size and battery life. The battery carries most of the weight in a Tablet PC. An in-built battery will last only around 2 hours but additional ‘booster’ batteries can be fitted which will extend battery life to up to 7 hours but also adds weight. To recharge the battery the unit must be plugged into a mains terminal. Due to the type of battery, to recharge 85% of its life takes approximately 2 hours. To recharge the last 15% takes another 2 hours. (approximately 4 hours in total). In almost all cases tablets can be attached to keyboard to perform as a desktop.

Tablet PCs come with a range of screen sizes (A5 up to A4 size). They also can be fitted with accessories, such as straps, to aid in their use for long periods of time. They can also be set up to use as desktop PCs.

When in the tablet form, a special ‘light pen’ is used to operate the device. When typing words or numbers, a device will usually have handwriting recognition software available as well as pop-up virtual keyboard which can be used to enter characters. ONS have concerns over how elderly interviewers will be able to operate a light pen – a training need, and also how respondents would react to the light pen as part of any self-completion modules.

4.4 Requirements

In deciding what we thought would be good hardware to trial; firstly, we took advice from other organisations (see above). Secondly, we asked some IPS interviewers what their requirements are. Lastly we had a set of office requirements:

1) Weight:

Weight is a very large consideration for any chosen device. A typical non-stop IPS shift would be up to two hours. Ideally the weight should be less than 1.6 kg.

2) Screen size

The bigger the better! (Given other constraints such as weight) The smallest would be around 18cm x 12cm. We can trial smaller screen sizes and need to investigate screen layouts and font sizes; e.g. one question per screen versus multiple questions per screen.

3) Robustness

These devices will be used in the field and must be fairly robust. They must be heat and rain proof, scratch resistant and anti-glare. Any device would require a strap and case.

4) Price

ONS would like to pay no more than £800 per unit (approx \$1500)

5) Hard Disk

Minimum 40GB

6) Battery Life

An IPS shift lasts two hours before a break followed by another two hour shift. A household interview for some ONS household surveys may take upwards of three hours. At least four hours standard battery life is required.

7) Minimum System requirements:

Blaise 4.80 requires Windows 98 SE or higher, Windows NT 4.0 or higher, a Pentium III or better processor, and a minimum of 64Mb of system memory (128Mb recommended).

4.5 Short listed options

Due to the very small screen size of the PDA and the weight of the hybrid PC, it was decided to trial a tablet PC. Below is a short-listed set of devices which were considered by ONS.

	1. HP Compaq TC4400	2. Motion LS800	3. Motion LS1600	4. Motion M1400
Dimensions (WDH)	285 x 244 x 44mm	227 x 170 x 22mm	296 x 245 x 18.7mm	296 x 240 x 22mm
Weight	2.2kg	2.2lbs	3.1lbs standard battery 4.1lbs extended battery	1.2kg
Hard disk	60GB	30GB or 60GB	30GB or 60GB	20 GB
Processor	2.0GHz Intel Core Duo	1.2GHz Intel Pentium M	1.6GHz Intel Pentium M	1.1GHz Intel Pentium M
Screen display	1024 x 768 pixels	8.4" display	12.1" display	12.1" display
Operating System	Windows XP Tablet Edition	Windows XP Tablet Edition (Vista ready)	Windows XP Tablet Edition	
Battery life	232 mins	29 WHr standard battery 57.7 WHr with extended battery	38.5 WHr standard battery 78.5 WHr with extended battery	263 mins
Other comments	Laptop mode			USB ports VGA connector Networking ports
Price	£999	From £1034	From £1092	£1300

	5. Lenovo T60	6. Fujitsu Siemens ST503x	7. Fujitsu Siemens ST5112	8. Psion iX104C
Dimensions (WDH)	27.4 x 3.3 x 26.7mm	324 x 220 x 24.9mm	324 x 220 x 24.9mm	284.5 x 209.6 x 40.6mm
Weight	1.71kg	1.6kg	1.6kg	2kg
Hard disk	80GB	40/60/80GB	60/80/120GB	40/80 GB
Processor	1.66GHz Intel Core Duo	1.2GHz Intel Pentium M	1.2GHz Intel Core Duo	1.1GHz Pentium M
Screen display	12.1" display	12.1" display	12.1" display	
Operating System	Windows XP Tablet Edition	Windows XP Tablet Edition	Windows XP Tablet Edition Windows Vista	Windows XP Tablet Edition
Battery life		6.5 or 10 hours	6.5 or 10 hours	3 to 5 hours
Other comments	Laptop mode Touch screen display Bult in modem	Built in modem USB ports Networking ports	Built in modem USB ports Fingerprint sensor	USB ports
Price	£1660			

5. Trialling Blaise on different devices

Due to short amount of time between instigating this project and writing this paper, at the time of writing this only one device has been trialled, although we plan more in the future. The device trialled was a Fujitsu Siemens ST503x. In addition to this trail industry representatives visited ONS to demonstrate hardware.

ONS currently uses Blaise 4.6 as its production version of Blaise. We trialled adapting a current Blaise 4.6 questionnaire for use on this device followed by a trial using the BASIL tool – part of Blaise 4.8.

5.1 Using existing Blaise 4.6 questionnaires

The first trial involved taking an existing survey questionnaire and running it on the tablet with no changes to screen layout or adapting answer options. ONS interviewers currently have the mouse activated which can be used for features such as the date picker. This feature is easy to use on a tablet using a light pen.

The majority of the questionnaire worked well on the tablet. Enumerated types are simple to answer and it is straightforward to move onto the next question. In landscape the questionnaire looks exactly as it does on the usual laptop. Problems arose if the screen was changed to portrait (this can be switched off). This resulted in some questions looking bunched and untidy.

A couple of issues with using the Blaise questionnaire in its exact form were with external file look-ups and String answers. If a look-up table did not find the correct answer in the look up at a first guess, it might be difficult, and time consuming to find the correct answer. The interviewer has an option to either scroll through the look-up to find the right answer, or re-enter the text to be used as a look-up to find a match. Both of these were not straightforward with the light pen. Similar to the look-ups, interviewers

require training for how to quickly and effectively use tablet keyboard with the light pen or alternatively activate and use the character recognition tool. If these are not mastered it could add to interview length. Although these are flagged here as issues, they can be overcome with interviewer training.

Another potential difficulty is using the ONS Question by Question (QbyQ) help. This interviewer aid can be used by pressing F12 on the key pad (or an option with the light pen). Once it is available, it is hard for interviewers to use the light pen to navigate the help and find specific sections. This is due to the small font used in the QbyQ help and the number of options, or 'button presses' to reach the desired help page.

5.2 Adapting 4.6 to run on a tablet

Due to the small number of problems encountered when running the Blaise 4.6 questionnaire on the tablet, only a very small number of changes were required to layout. There are instances where rather than having String answers, enumerated answer categories could be derived but, it is more the case that different interviewer training is required to familiarize interviewers with the practice of using light pens. If necessary radio buttons could have been enlarged (as long as the question could remain on one screen) but generally using the large screen question text size was not an issue.

5.3 Using Blaise 4.8

The release of Blaise 4.8 gave us an opportunity to test BASIL on a tablet. Although BASIL is essentially a tool for the internet it was thought that it could also be used for a social survey on a tablet due to BASIL's greater flexibility in terms of layout. We first ran the Statistics Netherlands supplied example BASIL questionnaire to get a feel of how the questionnaire worked on a tablet. We were very impressed with how easy it was to navigate and the freedom with which it is possible to customise the questionnaire.

The next step was to create our own BASIL questionnaire. First attempts at this were made by adapting the Statistics Netherlands code. Unfortunately this did not work for us without having the underlying understanding of how BASIL worked. In this case we started building a BASIL questionnaire from scratch and have now developed a short questionnaire in BASIL for use on a tablet. This questionnaire worked very well on the tablet. Visually the questionnaire looks good and extra features such as the menu facility allows for easy navigation around the questionnaire. Other advantages of BASIL are the easy 'next page' and 'previous page' buttons and obvious buttons to access question help.

BASIL is useful for short social surveys, as in this trail but the programming is longer than for standard Blaise. It would therefore be a heavy investment to upgrade all ONS social surveys to use BASIL. It would not be preferable to have surveys using different methods of data collection on the field.

6. Interviewer Training

HQ field staff were consulted when choosing a device to trial. These field representatives highlighted two main areas to test when deciding if a device was suitable for use in the field: i) interviewers ability to enter data given the potentially different screen sizes and layout of questions and ii) interviewers ability to use the device for long

periods of time while standing up – i.e weight and use of accessories. Plus the ease with which the device can be adapted to be used as a PC for home use as in administration data or coding.

6.1 Entering data into the device

At ONS we try to avoid the need to have any screen scrolling during a Blaise interview. It is therefore a need for the screen size of any instrument to be large enough to hold the information in a font size that is easily readable for the interviewer.

After ruling out the very small screens used on PDAs we looked at the Motion LS800 (227mm x 170mm), about A5 size and the normal laptop sized tablet (296 x 245). Although the LS800 is lighter and easier for an interviewer to carry and handle, it was felt by the ONS field managers that the font size is too small. It was decided that a larger screen size is required for any mobile device.

It is accepted that if ONS were to upgrade interviewers laptops to use tablet PCs training would be required for all interviewers. The main area of testing will be around the use of a light pen as opposed to a keyboard operated device.

6.2 Carrying the device and interviewing

Two large concerns with the interviewer's ability to use the device were the weight and battery life. Interviewers are required to interview for a two hour IPS shift (with breaks) or, if necessary, for a 2+ hour interview. A tablet on its own does not weigh around 1.6kg. Most devices will have a standard, built in battery with a life of 2 – 4 hours. It is possible to have add-on batteries which will make the device last longer in the field but will also add on weight. With some devices the weight can be disproportionate on the device and can make handling it very difficult. The Motion tablets however have a flat battery which maintains equal weight distribution across the device.

There are a range of accessories available to help hold tablet PCs when being used in the field. These aids include hand and arm straps which help to not only grasp the instrument but also to distribute the weight across the person to make the device seem lighter. Despite these accessories, there is a risk that using the device for long periods of time could result in a lot of strain for interviewers, especially around the neck and back areas. Any use of a tablet device would require a full-scale field test. Tablets would be useful for short door step interviews and household interviews where the interviewer can sit and rest the device but perhaps not for IPS interviewer shifts.

7. Conclusion

Different devices are more suited to different types of survey. A smaller hand-held device such as a PDA may be of use to an IPS interviewer in that an IPS interviewer works long shifts standing up and carrying out a short interview. However, the IPS interview contains large text answers and large look-ups (such as country travelling from \ to) which do not suit the small screen of a hand-held device. Other considerations are the potential increase to the length of the interview. IPS interviewers can very swiftly carry out and interview on paper and make notes in pencil to be written up later on. This is not possible using a light pen.

Household surveys are more suited to the larger tablet type PC which can also be used for doorstep surveys and respondent selection. There are, however, security issues with openly carrying expensive hardware to peoples households as opposed to a laptop in a bag or laptop case.

There is very little pressure from interviewers to change our current hardware. The main reason is on cost grounds for IPS interviewing. At present there is very little doorstep interviewing carried out which require a device to be used on the doorstep.

In addition to operating Blaise on the tablet, other aspects of the survey require testing. Interviewers currently manage cases and carry out pay claims via the laptop. These elements will also require testing on a tablet PC.

This paper investigates very early ONS trials at using devices other than laptops. A lot more investigation is required as to how they will operate in the field and how interviewers will cope on the different surveys. One conclusion we can draw is that Blaise is flexible enough to be used on a tablet PC.

8. References

David Hill, Deploying Blaise to Tablet PCs for Mobile Use, 2006

Michael W Gerling, A new look into portable electronic devices for field data collection in the National Agricultural Statistics Service

Using the Blaise® Alien Router For Audio-Recording of In-Person Interviews

M. Rita Thissen, Sridevi Sattaluri and Lilia Filippenko, RTI International

1. Introduction

The Blaise alien router is a useful technique for managing calls to external software. In this paper, we describe use of an alien router for activation of audio-recording software for in-person surveys.

During in-person field interviews, the performance of field staff and the effectiveness of item wording are notoriously hard to monitor. Traditional methods of live observation during the interview can bias results through the presence of the observer, and verification call-backs to the respondent to confirm authenticity and professional behavior may not provide enough detail to assure high quality data collection. For computer-assisted personal interviewing (CAPI), digital audio recording offers an efficient alternative which allows evaluation of interviewer and item performance during data collection (Thissen et al, 2007).

We describe a practical technique in which Blaise (Statistics Netherlands) instruments can be enabled for computer audio-recorded interviewing (CARI). This approach comprises an external file for specifying items to be recorded, an alien router which activates the laptop's sound-recording system, and block-level activation of the router within a Blaise instrument. Ideally, a system for audio recording should require little programming, execute undetectably, provide a flexible choice of which items to record, allow respondent consent to be given or revoked at any time and store recordings in clearly identified files. This system offers all of those advantages. By invoking the router on specific blocks or on the entire instrument, minor programming effort can turn a standard implementation into a CARI-enabled questionnaire.

2. The Blaise Alien Router

Blaise software provides ways to call external procedures, including a technique termed the “alien router.” Before and after each question is asked on-screen, a process known as the router executes instructions regarding layout, checks and other operations which are not apparent to the interviewer. Custom actions can be added to this process, by creating an alien router which runs in addition to (or instead of) the native router to “ask” the question. The alien router can communicate both with the instrument and with external software, such as executables or Windows components. The Blaise Component Pack (BCP) is required for execution of an alien router. ActiveX has been supported since BCP version 2 and Blaise 4.6. Additional information and examples are provided as part of the documentation in the Blaise Online Assistant.

3. Computer Audio-Recorded Interviewing (CARI)

Survey researchers have discovered value in computer audio-recorded interviewing (CARI) (Biemer et al, 2000). This approach helps improve the quality of data collection through unobtrusive recording of the audio portion of in-person interviews (Thissen and Rodriguez, 2004) or telephone interviews (Suresh, 2005); much as silent monitoring has

been used to ensure quality at many call centers. With CARI technology and consent from the respondent, sound files are recorded during the interview, transmitted to a central site and monitored for proper performance of the survey protocol.

Although there are several advantages offered by implementing CARI, perhaps the most compelling reason is to confirm the authenticity of data at lower cost than traditional verification methods. CARI can act both as a deterrent to curbstoning and as a tool for detecting questionable interviews. Informal feedback suggests that interviewers who are aware that monitors may listen to parts of each interview are less likely to falsify data, because the audio file acts as a “witness” to their actions. In this way, the simple presence of CARI can reduce cheating.

While evaluating authenticity, monitors may also rate interviewer performance for feedback. Performance characteristics may be positive or negative, resulting in praise or prompting additional training. Such feedback is especially important in the first few weeks of a novice interviewer’s work, at the start of a new survey, or when conditions for obtaining the interviews are known to be challenging.

CARI can be used to evaluate the usability of questionnaire items. The audio recording of an interviewer’s presentation of an item and the subject’s response provides a clear indication of whether the item is phrased in a way as to be read easily by the interviewer and understood easily by the respondent. Survey items which evoke negative reactions or require frequent explanation are detrimental to the response rate and increase the level of burden on both interviewer and subject. Using CARI, especially during field testing of an instrument, allows the survey specialist to evaluate the effectiveness of the questions.

A complete CARI process includes several steps. Recordings are made in Windows wave format and later compressed to mp3 format to reduce the size before transmission. Automated case management processes transmit the files, record their arrival in a database and present them for review through an interactive interface called the CARI Monitoring system.

1. Recording during the interview
2. Compression and encryption (optional)
3. Transmission and storage
4. File tracking and management
5. Monitoring and evaluation

This article discusses the first step of the process, collecting the audio recordings as part of questionnaire instrumentation.

4. CARI System

Audio recording can be added to Blaise instruments by using either of two programming approaches. One approach uses a Blaise procedure which in turn invokes an external application to start and stop the recorder. Using this approach requires custom programming within Blaise in every place the recording application needs to be invoked, with fields that keep track of whether recording is already in progress or needs to be started or stopped (Thissen and Rodriguez, IBUC 2004). The second approach is described here, using an external item list and an alien router instead of custom rules and

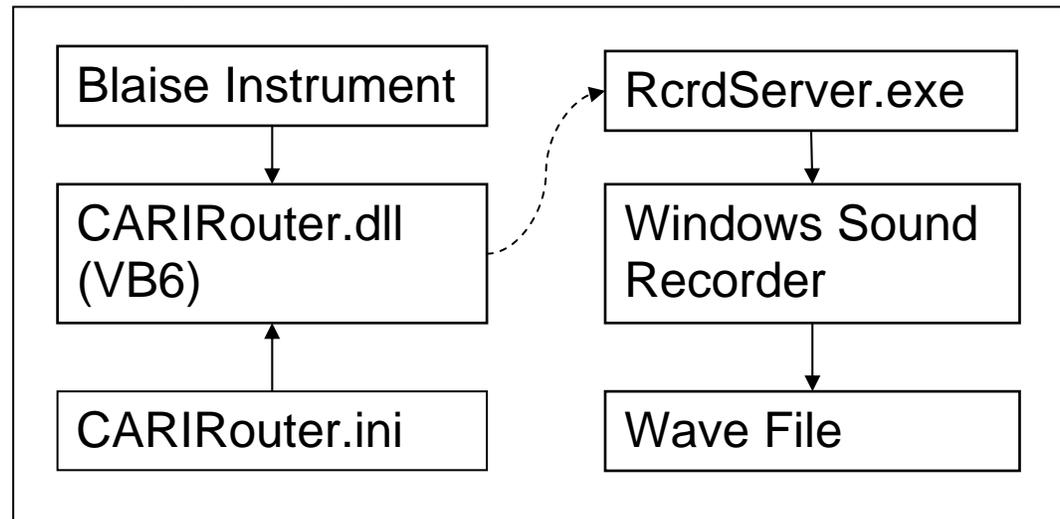
an alien procedure. In the second approach, instrument modifications for CARI are much simpler once the router has been implemented, because the same router may be reused for multiple surveys.

As with any programming challenge, there are several ways to meet the requirements. At RTI, the CARI alien router implementation consists of 5 parts:

- CARI Alien Router (CARIRouter.dll)
- RcrdServer.exe – a standalone executable that starts and stops recording by communicating with the Windows Sound Recorder.
- Blaise instrument – with block-level activation of the alien router
- CARIRouter.ini – an external text file that lists the items to be recorded

By using the native Windows Sound Recorder, performance is optimized during use and drag on processing is minimized. This approach is illustrated in Figure 1.

Figure 1. RTI's approach, using an alien router for CARI



4.1. The CARI Alien Router and RcrdServer

Blaise alien router technology allows the invocation of an external component before every survey item is given focus and after it loses focus. Use of the alien router externalizes the complexities of tracking the recorder state and offers the option of maintaining a text list of items to be recorded, external to the instrument. This reduces the complexity of instrument programming and allows easy modification of the list of items to be recorded, without any need to modify the data model or recompile the instrument (Thissen and Sattaluri, 2006).

Our goals were to implement an alien router which would do the following:

- Successfully start and stop the sound recorder from Blaise
- Successfully create audio files with the specified items
- Provide no indication onscreen that recording was starting or stopping.

- Decrease the programming effort as compared to using an alien procedure
- Provide a flexible method for CARI item selection

In our implementation, CARIRouter.dll is an ActiveX component, implemented in Visual Basic 6.0, which combines information passed in from the Blaise instrument about the current item and the CARIRouter.ini file to determine when to start and stop the recorder. It sends the start and stop commands to the RcrdServer.exe which in turn activates the Windows Sound Recorder. The router contains two functions (ManageCARI, StartCARI.), a reference to the Blaise API, and a form for communication with the sound recording system. Communication with RcrdServer takes place through Windows Sockets. As an alternative, the router could be programmed to directly control Windows Sound Recorder.

Figure 2. The ManageCARI function in CARIRouter.dll.

```
Public Sub ManageCARI (DB As BIAP14A2.Database, DS As BIAP14A2.DepState)

On Error GoTo Err_ManageCARI
strBlaiseDBPath = DB.DataFileName
strBlaiseDBPath = Mid(strBlaiseDBPath, 1, InStrRev(strBlaiseDBPath, "\") - 1)

InitializeRouter strBlaiseDBPath

consentflag = (DB.Field("CARIConsent").Text = "YES")
CaseID = DB.KeyValue
If consentflag Then
    'Get hold of the currently active question
    'The properties of the DepState parameter need to be used for this
    Set Layout = DB.Screens.LayoutSetCollection(DS.LayoutSetIndex)
    Set Parallel = Layout.ParallelCollection(DS.ParallelIndex)
    Set Page = Parallel.StoredPageCollection(DS.StoredPageIndex)
    Set Quest = Page.QuestionCollection(DS.QuestionIndex)
    Dim QID As String
    QID = Quest.Field.Name
    ...
    If DS.AlienRouterStatus = blrsPreEdit Then
        If bRecordEntireSurvey Then
            If Not CARIStarted Then
                StartCARI CaseID, QID, trig.Timer
            End If
        Else
            ...
        End If
    ...
End If
...
End If
...
End sub
```

The Blaise Instrument calls a function, “ManageCARI,” within CARIRouter.dll before and after the presentation of each item. The CARIRouter loads up the list of items from the CARIRouter.ini file when it is called for the first time. During every call to the CARIRouter, the Blaise instrument provides information to the router as to which item is

next on the path and if execution is before or after the presentation of the question item. The CARIRouter then matches the item name against its list of items to determine if it needs to be recorded or not. The router then communicates with the RcrdServer.exe to either start or stop the recording.

For the alien router to work in an environment where multiple Blaise databases may be present, the ManageCARI function has been defined to accept the Blaise database name and DepState objects as parameters. The code sample in Figure 2 shows part of the ManageCARI function to demonstrate the interaction between the Alien router and the Blaise Instrument.

The DB.FileName property holds the location of the Blaise database, which is used in turn to determine the location of the INI file. The InitializeRouter function reads in the values from the CARIRouter.ini file and initializes the connection to the RcrdServer.exe. Code inside the InitializeRouter function makes sure that the initialization happens only once. Examining the code in Figure 2 reveals several important actions:

- The parameters DB and DS are passed in by the Blaise Instrument and they identify the Blaise database and its current status.
- The DB.Field method can be used to retrieve the current field value of any Blaise database field.
- The Page.QuestionCollection(DS.QuestionIndex) provides the reference to the current question in the instrument
- The DS.AlienRouterStatus provides the information about whether the function was called before or after the presentation of the question item.

Figure 3 shows code from the StartCARI function. This function uses a flag, bOverwriteWavfiles, that is initialized from the CARIRouter.ini to determine if previous recordings with the same question name will be overwritten or not. If the flag is set to “yes” within the ini file, the contents of the recording file will be overwritten each time the question is asked. This may be desirable, in a situation where each a question is asked multiple times and only the final response should be kept. On the other hand, it may be desirable to set the flag to “no” if the respondent is allowed to back up through an item and pass through it again, because on the later passage the question may only be displayed briefly in transit, without any response. This sequence of actions may happen when a respondent or interviewer backs up through several questions to view the answer of an earlier question, and then uses arrow keys to move forward to the last unanswered question.

One of the alien router’s actions is to signal the recording software to start or stop, by communication with RcrdServer. Signaling is handled through a function SendRequest in the form frmRcrdClient. The first parameter of SendRequest determines that the recording needs to be started. The second parameter defines the timer value if any. The third is the name of the wav file to create. At RTI, we use a long file name, created with case ID and item name as part of the file name, to allow unambiguous identification.

Because the user may navigate back and forward through the instrument, possibly moving from items designated for recording to blocks or questions that should not be recorded, logic is required to keep track of whether a recording is already in progress. Similarly, if the respondent changes their consent from “yes” to “no,” the software must

be aware of the current state of recording. The `mbCARISStarted` and `CurrentTimer` variables are used to keep the status of whether a recording is currently in progress, to allow it to be stopped or started appropriately. The `StartCARI` function is used to start the recording, and the `StopCARI` function is used to stop it.

Figure 3. The `StartCARI` function in `CARIRouter.dll`.

```
Private Function StartCARI(CaseID As String, QID As String, Optional Timer As Integer = 0) As Long
    Dim s As String
    If bOverwriteWavfiles Then
        mStrSndFileName = CaseID & QID
    Else
        mStrSndFileName = CaseID & QID & Format(Now(), "mmddyyyyhhnnss")
    End If
    frmRcrdClient.SendRequest "1," & Timer & "," & mStrSndFileName
    mbCARISStarted = True
    If Timer > 0 Then
        CurrentTimer = Timer
        StartTime = Now()
    End If
End Function
```

Within the alien router, `frmRcrdClient` communicates with `RcrdServer` by using the `WinSock ActiveX` control. It initializes the `WinSock` control to use a particular port, to which the `RcrdServer` listens. The `SendRequest` function sends the request using the `SendData` method of the `WinSock` control. Requests to begin recording include a 1 to start recording or a 0 to stop recording. If the recording is to be made for a specific length of time, that timer value is also passed along. Finally, the router passes the file name to be used in saving the audio file.

For use with the Blaise instrument, `CARIRouter.dll` and `BlAPI4A2.dll` must be registered on the Windows laptop. Their location may be in the `System32` folder or elsewhere on the computer. In our application, `Microsoft WinSock.dll` must also be present and registered.

4.2. Enabling the Instrument for CARI

After implementing an alien router dll, the next step is to reference it and invoke it from the Blaise instrument. Within the instrument or block header, the presence of the alien router is indicated by the `ROUTER` statement:

```
ROUTER RouteRules ALIEN ('CARIRouter.CARIRecorder', 'ManageCARI').
```

The syntax of this statement, which alerts the DEP to the existence and identity of the alien router, can be found in Blaise Online Assistant documentation under the heading “Router.” In our implementation, it provides a name “RouteRules” which acts as the name of the alien router within the Blaise code. `CARIRouter` is the name of the dll itself, `CARIRecorder` is the public routine inside the dll, and `ManageCARI` is the name of the function which is activated.

Add the definition of the router to all blocks where CARI may need to start or stop recording. If CARI might be started in one block and deactivated in another, the router has to be associated with both blocks. The router can be associated with a top-level block, and it will affect all sub-blocks. Only one router can be active at a time, and so if

another alien router or external call has been defined in the instrument, block structure may need to be revised to keep the two isolated. See Section 5 below for additional comments about constraints on alien router usage.

The Blaise code requires some relatively simple changes to incorporate the CARI functionality. Every Block that contains questions that may be recorded should have the ROUTER ALIEN keywords as shown in the highlighted code in the following example.

Figure 4. Associating an alien router with a specific block

```
BLOCK Bperson
PARAMETERS
    {define parameters here...}
ROUTER RouteRules ALIEN ('CARIRouter.CARIRecorder', 'ManageCARI')
FIELDS
    {define Fields here...}
RULES
    {define logic here..}
ENDBLOCK
```

Refer to the Router in the Rules as shown in the highlighted code. The alien router will only be invoked if the router name is added to the block when it is called in the Rules. Figure 5 shows a simplified example of Fields definitions, including a question asking for consent to record the interview. In the subsequent Rules section, the block HHRoster has been enabled for CARI recording by adding “.RouteRules” to the call.

Figure 5. Calling a block with an associated alien router

```
{Locals and other definitions go here...}
FIELDS
    CARIConsent : tyesno
    HHroster : array[1..3] of Bperson
    {more fields go here ....}
    Q2 "Has consent been given?": bconsent

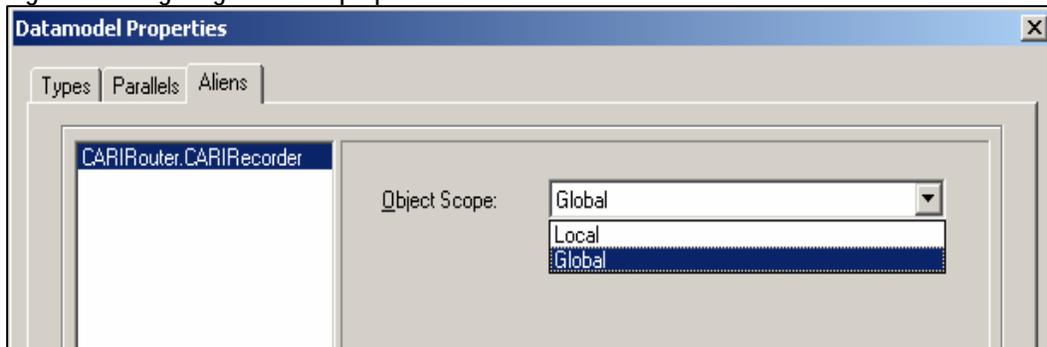
RULES
    {more rules go here...}
    CARIConsent.keep
    Q2
    For I := 1 to 3 do
        HHroster[I].RouteRules
    enddo
    {more rules go here...}
ENDMODEL
```

At RTI, we request consent at the start of the interview to permit recording, and we allow the respondent to change that response at any time during the session. For example, if a respondent initially refuses and later decides it would be acceptable, the CARIConsent item can be changed from “No” to “Yes.” Conversely, if a respondent initially assents, permission can be revoked by changing CARIConsent to a value of “No.” To manage revocable consent, define a Consent block that asks for the respondent consent as a routed parallel block. If multiple consents must be stored, as in a household interview, use a separate kept variable to hold the value of the current respondent’s CARIConsent.

Depending on the rules of the state or country in which the interview is conducted, it may or may not be legal to record the consent question itself.

There is one configuration item to set, for the CARI instrument to run well. Set Object Scope for the alien router in the Data Model Properties to Global so that a single instance of the router runs when the instrument runs, as shown in Figure 6. Otherwise, a new instance of the router will be created for each item, resulting in a drain on performance and memory.

Figure 6. Configuring the router properties



The CARIRouter handles instrument navigation anomalies such as backing up or breaking off while recording is taking place, instrument logic complexities such as looping and skips which may need to start and stop the recorder repeatedly, and as mentioned above the option of revocable consent.

4.3. Specifying Items to be Recorded

At this point, the CARIRouter has been defined and the Blaise instrument has been adapted to make use of it. Block-level association in the Rules invokes the alien router. How does CARIRouter know when to turn the recorder on or off? At RTI, that information is contained in an initialization (ini) file external to both the router and the instrument. There are many possible ways to communicate the CARI item list to the router, and we chose this approach for its flexibility and ease of use.

At RTI, the CARI ini file is a simple external text file for specifying items to be recorded. Our ini scheme offers great flexibility in selecting portions of the interview for recording, including:

- Single items as individual files
- Sequential items in a single file
- Timed recordings
- Starting a new recording when an earlier one is still in progress
- Recording an entire survey into a single file
- Indicating specific elements from an array

Logic to control recording under these circumstances is specified in the ini file and executed by the alien router.

Because it will be parsed by the alien router, the ini file has a defined structure and syntax for the statements contained in it. The file must be named “CARIRouter.ini” and reside on the laptop in the same folder as the Blaise database. An example of a simple ini file is given in Figure 7, and its details are explained in subsequent paragraphs.

Any line in CARIRouter.ini which begins with a semi-colon as the first character or is completely blank will be ignored. This provides a convenient way to include comments or descriptive information. Each section starts with a header line, which is [Options] or [Items].

Figure 7. Contents of a simple CARIRouter.ini file

```
; CARIRouter.ini

[Options]
RecordEntireSurvey = No
OverwriteWavfiles = No

[Items]
; Specify the itemcount as the number of entries that follow.
ItemCount=4
; Record 20 seconds following the start of Q5, regardless of how many items are included
Item1=Q5, CONTINUE, 20 ; timed recording for 20 seconds
; Record items Q12 and Q13 as a single file
Item2=Q12, START
Item3=Q13, STOP
; Begin at the third element of Q22 in block DEMOG and record for 30 seconds
Item4=DEMOG.Q22[3], CONTINUE, 30
```

The Options section provides specifications that apply to the entire instrument. At this time, two options are supported, one to indicate whether recording is to be done for the entire instrument or only parts of it, and the other to indicate whether wave files should be given a unique time-stamped name so that the same item may be recorded multiple times in a case if needed for loops or other repetitive situations.

The Items section consists of 2 or more lines. There will always be a single ItemCount line, and there will be one or more Item lines after it. The Items section lists the questions which should be recorded, and for each item an “action” is specified. The available actions are

- START – begin recording when this question is presented
- STOP – end recording after this question’s response is entered
- CONTINUE. – if recording is already on, continue to record

Any question which is not listed in the Items section will not be recorded, unless it is administered within a timed period.

Sometimes it is desirable to record for a specified length of time, rather than indicating on which question to start and stop. For example, if the path branches after a CARI question, a timer can be set to terminate the recording, instead of attempting to specify a stopping point on each branch. Item 1, Q5, in Figure 7 shows how a timer can be set, in this case for 20 seconds.

An advantage of using an external file to hold the list of CARI items is that it is easy to modify without changing the Blaise instrument itself. Programming the questionnaire can progress independently of decisions about which items to record. Tests and debugging of the recording process can be performed with recordings on questions early in the questionnaire and easily reached. This approach even permits modification of the CARI item list after an instrument has been released to the field without returning the laptop for updates and without affecting the data model. A new list can be transmitted to the laptop, replacing the original CARIRouter.ini, with additional items to record or different settings.

5. Constraints

Although the alien router technology can be very useful, as in the case of activating sound recording for CARI, there are times when it is not compatible with other features of Blaise. For example, the CARI alien router supersedes any other alien router within a Blaise block. For instance, the alien router supersedes usage of “lookup” and “classify” methods, because creating these in Blaise makes use of the alien router technology. If a “lookup” table is present in the same block with a user-defined alien router, or in a nested block, the lookup table will not work correctly. The dialog box which presents a value from an external database or classification type is not loaded by the DEP.

To avoid such conflicts, it may be necessary to redefine the block structure to segregate the fields using an external database or classification type the CARI alien router or other user-defined router.

6. Summary

By using a Blaise alien router with an external list of instrument items to record, CARI can be implemented quickly and easily, with little programming effort beyond the original development. A few lines of Blaise code are required to associate an alien router with a block when converting an existing survey instrument to one with CARI enabled. A simple text file serves as an external list of questionnaire items to record, allowing survey managers to amend the list during development or even after production data collection begins, without any effect on the data model of the instrument.

This technique provides support for easier quality assurance techniques which reduce burden on the respondents, field staff and surveying organization.

7. Acknowledgements

The authors would like to recognize R. Suresh, Albert Bethke and Phil Cooley of RTI International as the inventors of CARI in 1999. We would also like to acknowledge the contributions of Paul P. Biemer, RTI International, for developing and evaluating CARI methodology, the U.S. Census Bureau for research contributions, and many members of the Research Computing Division, Education Studies Division and Survey Research Division of RTI who have helped with suggestions, encouragement, design ideas, testing and conquering obstacles.

8. References

- Biemer, P.P., Hergert, D., Morton, J. and Willis, W.(2000), “The Feasibility of Monitoring Field Interview Performance Using Computer Audio Recorded Interviewing (CARI)”, *Proceedings of the American Statistical Association’s Section on Survey Research Methods*, pp. 1068-1073
- Statistics Netherlands, Statistical Informatics Department, P.O. Box 4000, 2270 JM Voorburg, The Netherlands.
- Suresh, R. (2005). “Web-Based Computer Audio Recorded Interview (Web-CARI).” Presented at the *International Field Directors and Technology Conference 2005*, Atlanta, GA
- Thissen, M. R., and Rodriguez, G. (2004), “Recording Interview Sound Bites Through Blaise Instruments”, *Proceedings of the International Blaise Users’ Conference*, pp. 411-423.
- Thissen, M.R, and Sattaluri, S. (2006), “Research and Development in Audio-Recorded Interviewing, Part II”, Presented at *The International Field Directors and Technologies Conference*, Montreal, Canada
- M. Rita Thissen, Sridevi Sattaluri, Emily McFarlane, and Paul P. Biemer (2007), “Evolution of Audio Recording in Field Surveys”, Presented at *The 2007 Conference of the American Association of Public Opinion Research*, Anaheim, CA

Dynamic List Building Using Blaise

Barbara S. Bibb & Gilbert Rodriguez, RTI International

1. INTRODUCTION

Data collection instruments frequently ask redundant questions that collect data containing the same or similar information. For instance, enumeration questions might collect addresses for multiple people where each person may have the same address, or might have a different or alternate address. This paper addresses the problems and issues of building a dynamic list of respondent answers and using that list as a ‘pick list’ for answers to future questions in the instrument for the administration of that same case.

To reduce both interviewer error entering the data as well as respondent burden repeating the same answers over and over again, Blaise can be programmed to build a list of suitable answers maintained within each case. For subsequent times through these questions, the instrument will display a list of the previously entered answers. If the next respondent answer is on the list, the interviewer can simply pick the appropriate entry. If the respondent offers a “new” answer, that answer will be entered as data and added to the evolving list. That answer will become available the next time a question is asked where that set of categories is appropriate. This paper will discuss the programming necessary to build such lists.

2. GENERAL BACKGROUND

A survey is an efficient tool for collecting data from various pools of respondents. For collecting some types of non-sensitive data, a survey administered by an interviewer yields more accurate data than a self-administered survey. To increase the benefits of interviewer-administered surveys, respondent burden and the burden on interviewers should be kept to a minimum. To reduce interviewer burden, the amount of effort to enter an answer into a CATI instrument should also be kept to a minimum.

Some data collection efforts attempt to collect repeated, open-ended data. In some cases, several of the questions can have the same or similar answers. String data, repeatedly collected, can be tedious to key. Anything tedious to key is also very error prone.

Data entry of string data that is keyed multiple times cannot be easily compared. Any variation in spacing, spelling, and capitalizations will cause the comparison for a match to fail. By putting such data on a pick list, each occurrence of the entry will be exactly the same. This will create a more consistent set of string entries for each data record.

Examples of such data:

- Addresses associated with each household member
- Names of schools attended by members of a household
- Names (Last)
- List of medications

- Services received by members of a household
- Phone numbers
- Company names

The goal of list building code is to create a list of string responses within each administration of the instrument. This list is used as a tool for the particular household and particular questions being administered. This list is displayed for each question using the required set of similar responses. The interviewer can add a new response or pick a previously recorded response to answer the current question. The previously entered responses will store the data the same way it was originally entered. There will not be any variation in the string response from question to question where the answers are the same as a previous answer.

For example: Roster the children of a household. Collect data for each child enumerating the names of the schools each child attended during early childhood and elementary school.

Child 1 attended Community Preschool, Mount Forest Elementary School and Bolin Creek Elementary School

Child 2 attended River Ridge Elementary School and Bolin Creek Elementary School

Child 3 attended Mount Forest Elementary School and River Ridge Elementary School.

The first time a question comes up asking for the school name the list would be empty. The only option would be to enter a “new school name”. The second time the question comes up, the first answer would follow the option to enter a “new school name”. The interviewer could either choose the name previously entered or enter a “new school name”. If the response is the previously entered name, the interviewer would save time and effort by choosing the answer rather than re-entering that same answer previously mentioned. The working list would be maintained by the program and additionally the correct responses are kept for each child.

3. SPECIFIC BACKGROUND

The particular code being discussed was developed by RTI International for the Bureau of the Census (BOC). It was implemented for the BOC QDERS (Questionnaire Design Experimental Research Survey) Project. For this survey, each household was rostered. Once the roster was complete a series of follow-up questions were asked about each household member and the possible alternate addresses they might have had. Data was stored on a household level and by person. Specific questions were asked to determine where each household member was actually living throughout the year. The answers to the address questions built the pick list for the subsequent questions. Multiple addresses occurred if any household member stayed any place other than the primary residence.

For example household members could have been away for a job, away at college, in the military, living in a joint custody situation, relocated, had second homes, etc.

In the following discussion and for the examples, all names, addresses, and other data elements are fictitious. They were inserted to provide realistic looking examples.

4. IMPLEMENTATION

4.1 Considerations

Before beginning to code, programming constructs were developed to handle the process of maintaining the address lists. There are numerous issues to consider before beginning. Thought was given to the block structure needed to handle the task. Considerations included:

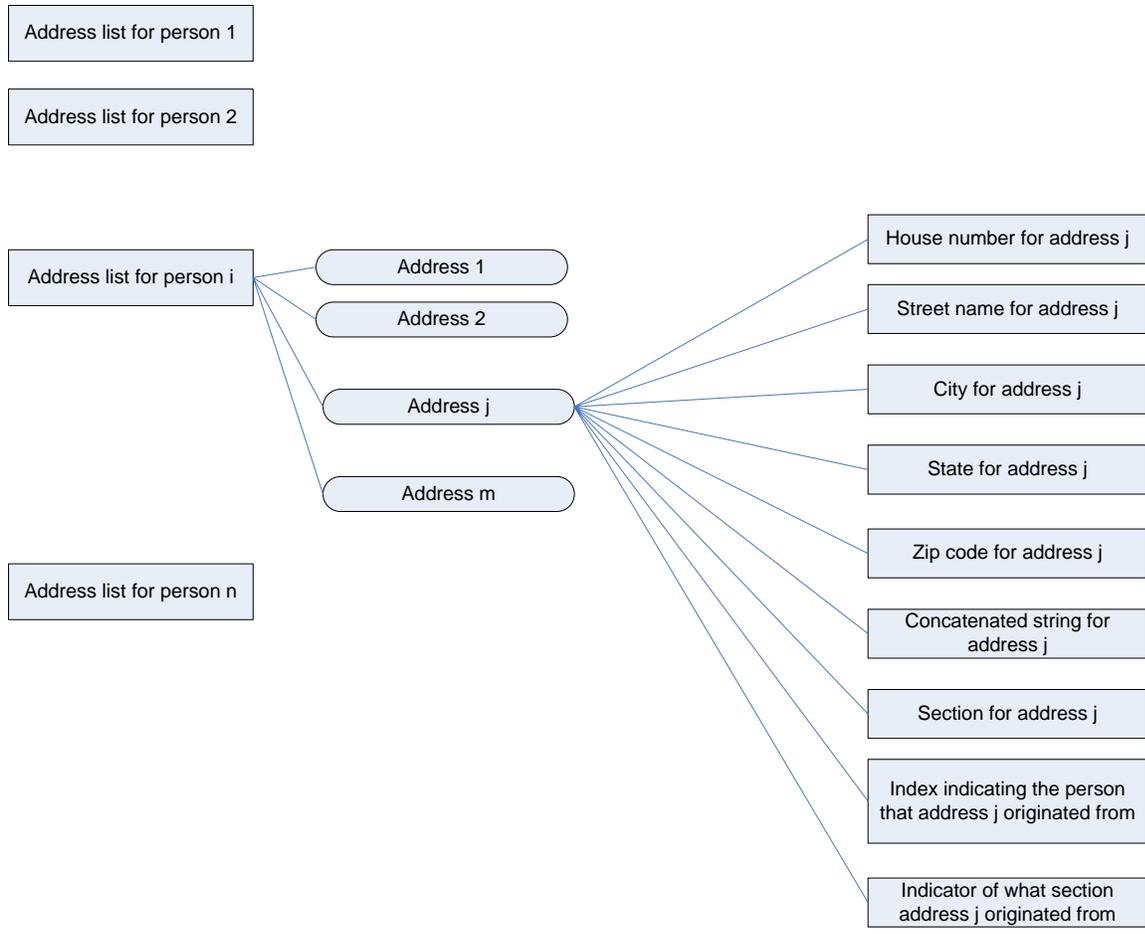
- the way the Blaise application re-evaluates the path throughout the interviewing process
- the number of points at which addresses could be added
- exceptions to the standard address collection block
- the handling of the counters to keep them in sync
- individual data storage for each household member
- global address collection array for unique addresses within the instrument

The complicated path structure and Blaise's variable assignment methods influenced the coding used to keep the lists of variables up to date and accurate. Counter variables can easily get out of sync, especially in a heavily nested block structure. Corresponding local variables were used to compensate for the possibility that the global variables might not remain accurate depending on the path through the instrument.

The standard address collection block was called using parameters. These parameters described where the data was coming from. Variables were kept so that the original source of each address could be tracked. These parameters decreased the reliance on global variables that could easily have gotten out of sync.

4.2 Coding design

A 1-dimensional block array indexed by the person in the household roster is defined at the top-level of the datamodel. Within the block array, array fields are defined corresponding to the different addresses for a particular person. These fields correspond to address number, street, state, zip code, and from which section of the datamodel the address originated. So, a list of addresses will be maintained for each roster member.



4.3 Implementation outline

The block `bhmemaddresses` tracks unique addresses for a particular person.

```

block bhmemaddresses
  parameters
    import persnum : integer

  fields
    numaddr : 0..15
    numaddrHN : array[1..15] of STRING[21]
    numaddrSTREET : array[1..15] of STRING[60]
    numaddrCITY : array[1..15] of STRING[20]
    numaddrSTATE : array[1..15] of STRING[3]
    numaddrZIP : array[1..15] of STRING[5]
    numaddrSTRING : array[1..15] of STRING[130] { concatenated address string }
    numaddrFIELD : array[1..15] of STRING[3] { indicator of where the address comes
from }

```

```
numaddrROSTNUM : array[1..15] of 1..9 { person }
numaddrROSTFIELD : array[1..15] of STRING[3]
```

```
auxfields
```

```
tnumaddrHN : array[1..15] of STRING[21]
tnumaddrSTREET : array[1..15] of STRING[60]
tnumaddrCITY : array[1..15] of STRING[20]
tnumaddrSTATE : array[1..15] of STRING[3]
tnumaddrZIP : array[1..15] of STRING[5]
tnumaddrFIELD : array[1..15] of STRING[3]
tnumaddrROSTNUM : array[1..15] of 1..9
tnumaddrROSTFIELD : array[1..15] of STRING[3]
tempaddresslist : array[1..15] of STRING[120]
```

```
rules
```

```
{ initialize temporary arrays to hold the address values for this person }
```

```
{ for each address block in the datamodel for this person }
```

```
if address block for this person is not empty then
  set temporary array elements corresponding to this address
endif
```

```
{ In the address list for this person, only keep unique addresses and store them in the
address array }
```

```
{ tally the number of unique addresses and set the number of unique addresses for
this person }
```

```
endblock
```

```
DEFINE FIELD BLOCK ARRAY OF ADDRESSES FOR EACH ROSTER MEMBER
```

```
memaddresses : array[MAX SIZE OF HOUSEHOLD] of bhmemaddresses
```

```
DEFINE GLOBAL FIELD AND ARRAYS FOR ALL UNIQUE ADDRESSES
THIS IS USED FOR DISPLAYING PREVIOUSLY-ENTERED ADDRESSES IN
ADDRESS QUESTIONS
```

```
numaddrall : 0..MAX TOTAL ADDRESSES
```

```
address_list : array[1..MAX TOTAL ADDRESSES] of string[120]
addrlist_hn : array[1.. MAX TOTAL ADDRESSES] of string[21]
addrlist_street : array[1.. MAX TOTAL ADDRESSES] of string[60]
addrlist_city : array[1.. MAX TOTAL ADDRESSES] of string[20]
addrlist_state : array[1.. MAX TOTAL ADDRESSES] of string[3]
addrlist_zip : array[1.. MAX TOTAL ADDRESSES] of string[5]
addrlist_rostnum : array[1.. MAX TOTAL ADDRESSES] of 1..9
addrlist_where : array[1.. MAX TOTAL ADDRESSES] of string[5]
```

IN THE RULES FOR THE MAIN BLOCK, PLACE THE FOLLOWING LOOP IN KEY LOCATIONS TO KEEP THE ADDRESS LISTS UP-TO-DATE.

```
for i := 1 to NUMBER OF ROSTER MEMBERS
do
  memaddresses[i](i)
  memaddresses[i].keep(i)
enddo
```

In addition to the addresses for each roster member, a global address field array is maintained that contains the unique addresses. Associated with this global address field array is a block type that displays these addresses for fields that require an address to be entered.

In a field requiring an address, the field will have a type with the first response choice being “New address”, and the remaining response choices showing the unique, previously-entered address strings, collected on the global address list. For the first potential address, the address list will be empty, so “New address” will be the only response option available.

The screenshot shows the QDERS v6.0 10/20/2006 interface. At the top, there is a menu bar with 'Forms', 'Answer', 'Navigate', 'Options', and 'Help'. Below the menu bar, there are several buttons: 'QDERS', 'F10', 'Rost', 'faq', and 'showroster'. The main content area has a yellow background and contains the following text:

What was your address on April 1st?

- If the respondent doesn't know the full address, probe for city, state or any other part of the address
- Do not include P.O. box address
- Press 1 if DK or REF, then enter DK or REF on next screen.

Below the instructions, there is a radio button next to the text '1. New address'.

Here is an example screenshot displaying some previously-entered addresses.

The screenshot shows a web application interface for QDERS v6.0 10/20/2006. The top navigation bar includes 'Forms', 'Answer', 'Navigate', 'Options', and 'Help'. Below this is a menu with 'QDERS', 'F10', 'Rost', 'faq', and 'showroster'. The main content area has a yellow background and contains the following text:

What is the address of the other place Silly Q James stayed?

- -If the respondent doesn't know the full address, probe for city, state or any other part of the address
- Do not include P.O. box address

- Press 1 for DK or REF, then enter Ctrl D or Ctrl R on next screen.

Below the question is a list of radio button options:

- 1. New address
- 2. 101 My Rd Hometown GA
- 3. 978 Oak Street Old Town FL
- 4. 67 Ivy Lane Old Town FL
- 5. 555 Sunset Way Grifton FL

If response category 1 (“New address”) is selected, then follow-up fields are displayed for entering the various address fields. Once the address fields are filled in, then the address is checked for uniqueness against the list of all addresses. If the new address is unique, then it is added to the global address list and the global address count is incremented. The field values linked to the new address are set to indicate which address question and for which roster member this address came from.

If an existing address is selected, then the fields for the current address are set equal to the corresponding fields of the existing address.

In the BOC QDERS project a similar approach was also used for dates. This concept can be adapted to work with any other commonly-used list of responses.

QDERS v4.0 9/05/2006

Forms Answer Navigate Options Help

QDERS F10 Rost faq showroster

What date did Willie Jones move to 2201 Rock Terrace Drive Austin TX?

1. New date

2. JAN-5-2007

3. MAR-10-2007

5. CONCLUSIONS AND SUMMARY

The purpose of developing a dynamic list is to reduce both respondent and interviewer burden. Reducing burden increases accuracy and provides cleaner back end data. The methodologists at the BOC asked that the dynamic list approach be implemented for their QDERS project, specifically for the collection of household addresses and move dates. This concept can be applied to any open-ended data collection effort where enumerated categories are not readily apparent. This approach is most appropriate for string data but can also be used for other data, like the QDER date data.

Some observations for the QDERS project:

1. Not all address questions are asked in exactly the same way, and oftentimes special case code had to be inserted in the address list maintenance based on the address question. For example, if an address was a military address, the datamodel had to be able to handle if the address was onboard a ship.
2. Code for displaying addresses had to be repeated in some locations within sub blocks so that newly-added addresses would consistently appear in the response category address lists. This is due to how the Blaise Selective Checking Mechanism works. At the block level, if a new address was added to the global address list, then there were situations where the address would not immediately appear in the list of addresses until a re-evaluation of the rules was performed.

3. If the user backed up, then you would only want addresses that were entered up to that point to appear in a response list. This was done by suppressing addresses from appearing if the question from which the address came from comes after the current address question.

4. Due to nesting of block arrays within other block arrays, processing of data required special effort.

6. ACKNOWLEDGEMENTS

Parts of the research upon which this report is based were supported by the U.S. Census Bureau, and we would like to acknowledge members of the BOC Questionnaire Design Experimental Research Survey team for their help. The views expressed are those of the authors and not necessarily those of the U.S. Census Bureau. This report is released to inform interested parties of ongoing research and to encourage discussion.

CTT: CAI Testing Tool

Mary Dascola, Genise Pattullo and Jeff Smith, The University of Michigan

1. Introduction

In 2006 Survey Research Operations (SRO), a unit within The University of Michigan's Survey Research Center, initiated the development of a tool for testing Computer-Assisted Interviewing (CAI) instruments. Without adequate tools for systematic testing of the instruments, a research study may experience production delays, interviewer frustration with instruments that don't function properly, or collection of imperfect or incomplete data that fails to meet the needs of the research project. Prior to the development of the tool, testing comments were recorded in an Excel spreadsheet or a Word document. After months of testing and iterative testing rounds these documents were cumbersome and time consuming to maintain.

To increase the overall quality of Computer-Assisted Interviewing (CAI) through standardized testing procedures, reduce the cost of testing, increase access to information concerning CAI development and software de-bugging through preset and ad hoc performance metrics reports, the Computer-Assisted Interviewing Testing Tool (CTT) was developed. The CTT is a comprehensive tool set that is used for testing and reviewing survey instruments and reporting on the testing process. The CTT application was developed to facilitate the testing of Blaise survey instruments by a variety of local and remote users. The CTT application automatically captures and consolidates testing comments, provides bug reports, assigns items to application developers and records testing outcomes.

This paper will discuss the major components of the CTT application which will include the Preload builder, Instrument testing, bug recording, random case generator and reporting on testing notes. The CTT Administration system will be reviewed as well.

2. Objective

In the past, research staff developing CAI applications did not have adequate tools for systematic testing of the instruments. As a result, projects would often have to send out revised data models during the production phase which added to the cost of administering each interview.

After careful review of current products on the market, we made an internal decision to develop our own CAI testing application. We wanted an application that would minimize the costs of modifying data models and sending updates to the field staff, and the costs of missing data callbacks to respondents. Secondly, standardization of testing across projects. Our testing components are generic and can be used by all projects. Each project prepares study specific preload and scripts that run through CTT to ensure testing of critical path scenarios. In addition, standard quality assurance checks are performed on the resulting data and ad hoc testing performance metric reports are available and used to guide future testing efforts.

CTT has increased the overall quality of CAI instruments through standardized testing procedures and reduced the cost of testing through automated regression testing and

random case generation. We were also able to facilitate development of risk-based testing plans and increase access to information concerning CAI development and software de-bugging through preset and ad hoc performance metrics reports.

CTT will facilitate testing at various stages of development (individual modules, sets of modules, the complete instrument), with or without a sample management system. Our system is adaptable for testing on multiple platforms, i.e., on the network, desktop, or interviewer laptop. Replaying previous test cases with new data models and creating preload strings for different testing scenarios will reveal past errors to project staff quickly. CTT automatically captures and consolidates testing comments and bug reports, reviewing and assigning items to programmers, and recording outcomes. Development of CTT was in C#.NET with Visual Basic .Net components with a SQL Server 2005 Database.

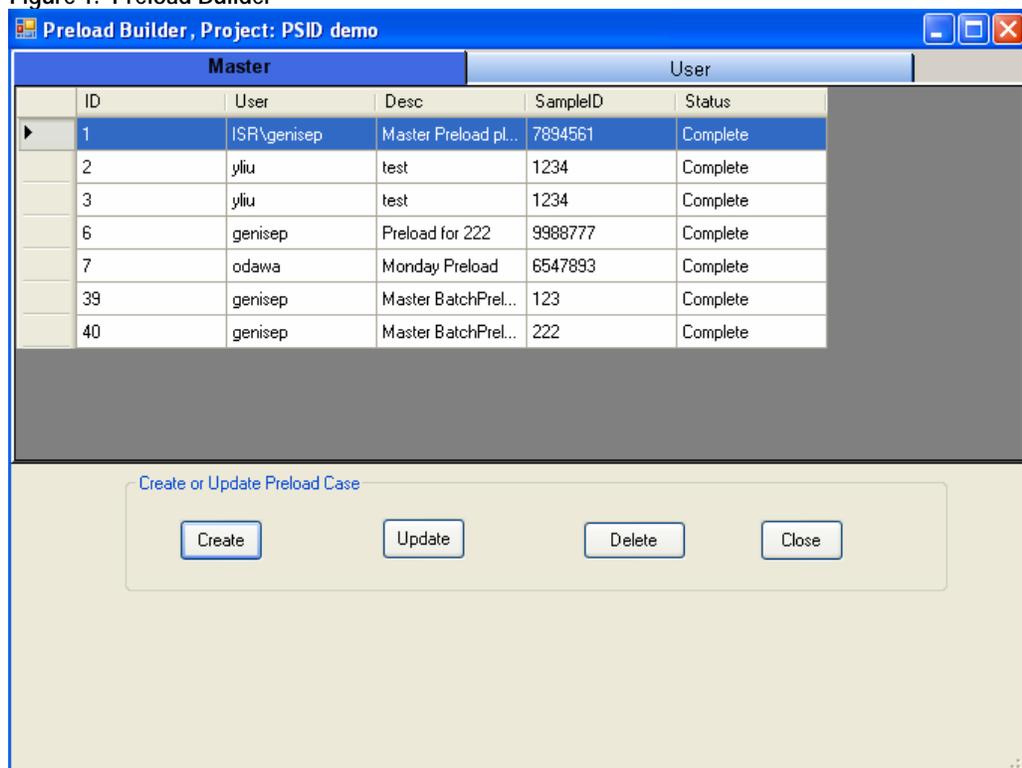
3. Components

The CAI Testing Tool includes six components for implementing and managing the testing process at various stages of instrument development. The Preload Builder and Instrument Testing modules are the main tools for actual testing of the instrument. Managing Problems and Reports facilitate maintenance of testing bugs found, fixed, or needing additional attention from the Testing Coordinator. The Random Case Generator tests skip patterns using data randomly generated by the system. The final module, Admin, manages administration of the entire testing tool, including adding projects for testing, assignment of testers to projects, and access rights.

3.1. Preload Builder

To begin the testing process, preload lines are needed that contain data appropriate to the instrument being tested; that is, data that closely resemble the actual sample that will be used for production interviewing. The Preload Builder allows a tester to enter data and create preload lines without having to wait for another staff member to add different scenarios. A bit of “intelligence” about the production sample and questionnaire is necessary when entering preload data. Following is the first screen you come to when you choose to create preload.

Figure 1. Preload Builder

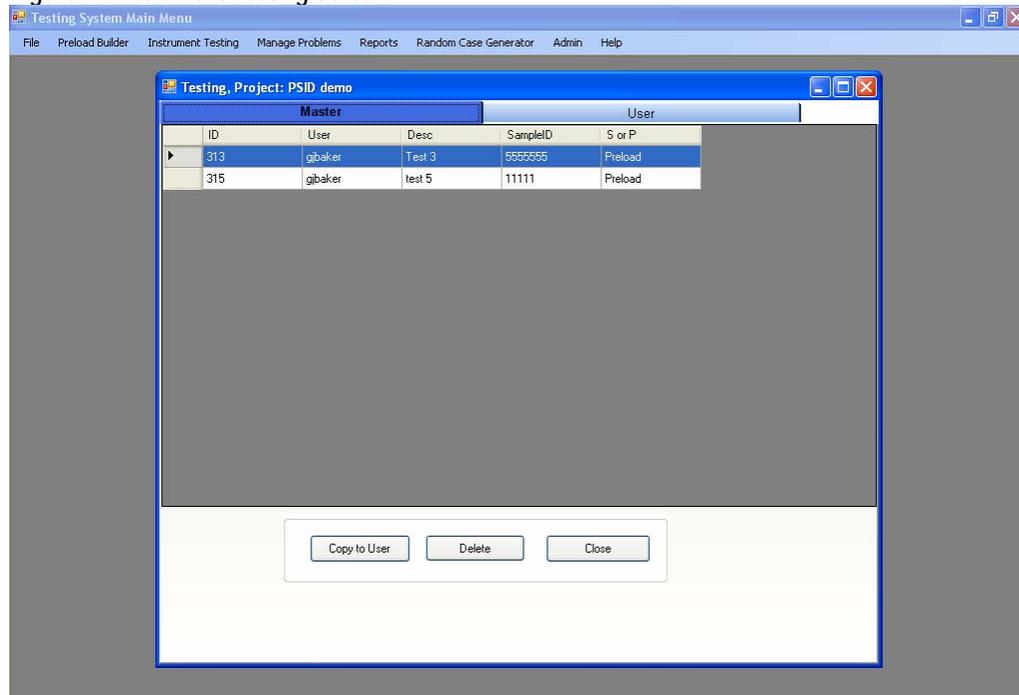


Once a preload line is completed, it can be updated any number of times. It can also be updated and saved as new preload, thus creating additional lines representing different sample situations. Preload lines can be saved to the User tab so that only the tester has access to the lines; or lines can be saved to the Master tab, making the lines available to all testers assigned to the project.

3.2. Instrument Testing

Preload lines created in the Preload Builder are presented for selection in the Instrument Testing module. A preload line is not altered in any way when it is selected for testing the instrument. It is possible, though, to create a “scenario” that captures all data a tester entered while testing the instrument. Using a preload line to create a scenario is akin to accessing a sample line and conducting an interview. It is not a requirement that a scenario is saved from every testing session. A tester can access a preload line, test the instrument, enter testing notes, and exit the instrument without saving. Following is the first screen you are presented when you select Instrument Testing.

Figure 2. Instrument Testing Screen



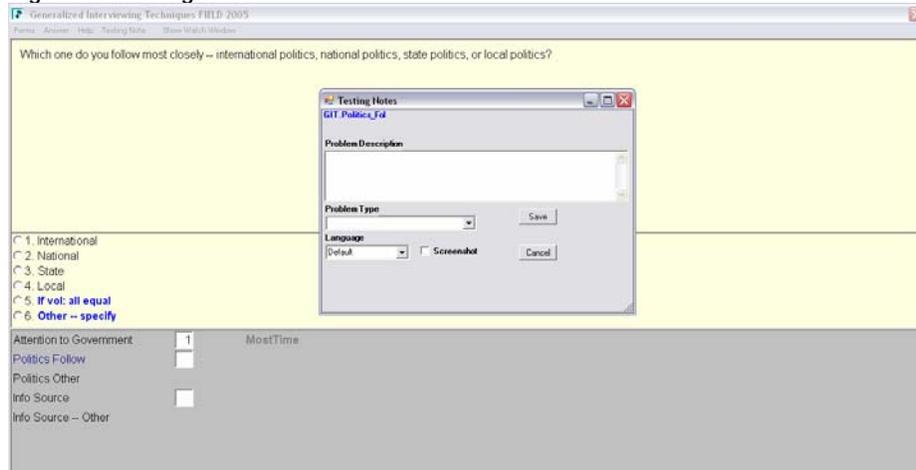
The Master tab allows you to view scenarios and preloads that have been determined to be useful, thus eliminating the need for each tester to make their own preload. If you wish to test a scenario or preload on the Master tab, you simply select the button “Copy to User” and that scenario/preload will be copied to the User tab.

Scenarios can be updated as many times as needed. A scenario can also be updated and saved as a new scenario, creating additional lines representing different interview situations. Scenarios are significantly useful when re-testing problems that were recorded in a previous testing session and fixed by the programmer. Scenarios are also important for testing revised data models. In both situations, using a scenario prevents a tester from having to re-answer questions in the instrument. A tester only needs to use the Jump to Field (shift F9) feature to get to the desired field for re-testing.

During testing, all problems are saved to a master database that records important information about each problem, such as the problem ID, the tester who created the Testing Note, the data model used, the section and field indicator, question text, and the description of the problem. Prior to CTT, if the tester made an error recording the question the problem occurred on, it would become difficult for the programmer to find the question the tester was talking about. This tool eliminates the confusion and wasted time.

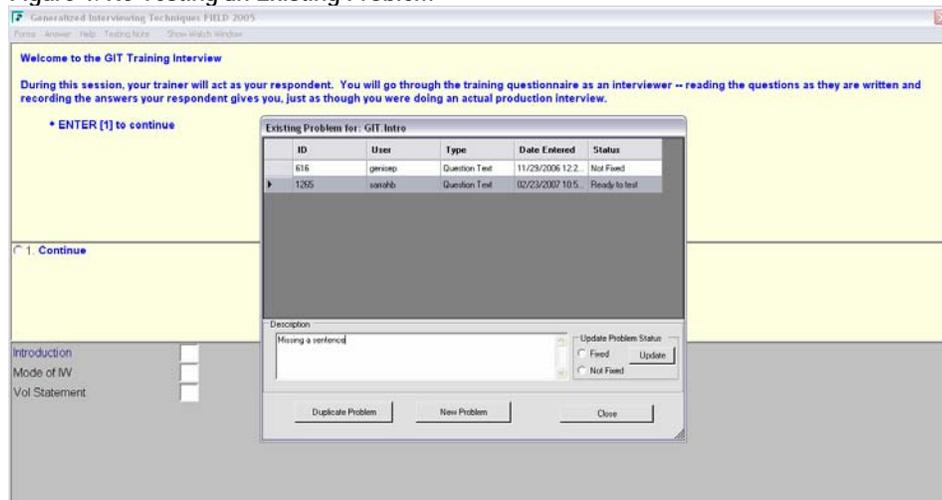
Pressing a single key, F8, will bring up the Testing Notes screen and allow the tester to easily record their comments and resume testing.

Figure 3. Entering a New Problem



If a bug or change has been requested already for a question, a list of the reported issues will be displayed for the tester. This feature has proved to be a big time saver since the tester can review previously entered comments, thus reducing the number of duplicate entries.

Figure 4. Re-Testing an Existing Problem



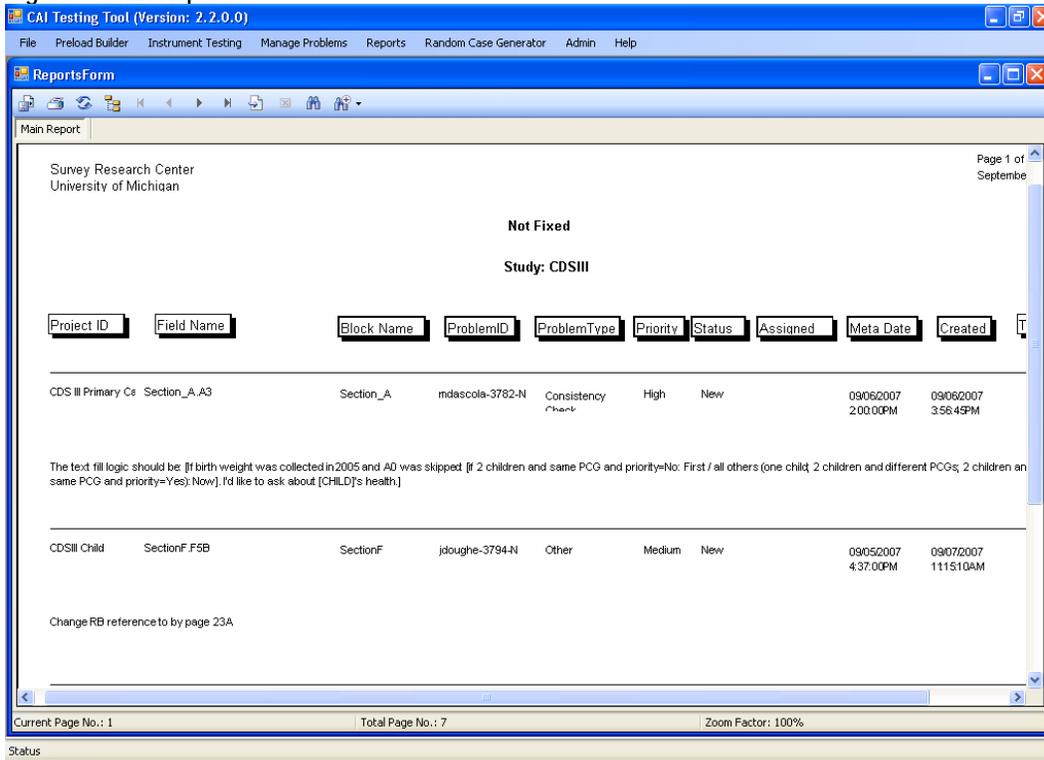
3.3. Reports

Reports are available to assist in the management of the testing process. When the testing round has been completed, the programmer will generate the “Not Fixed by Programmer” Detail Report from the Reports menu.

Figure 5. Available Reports



Figure 6. Main Report

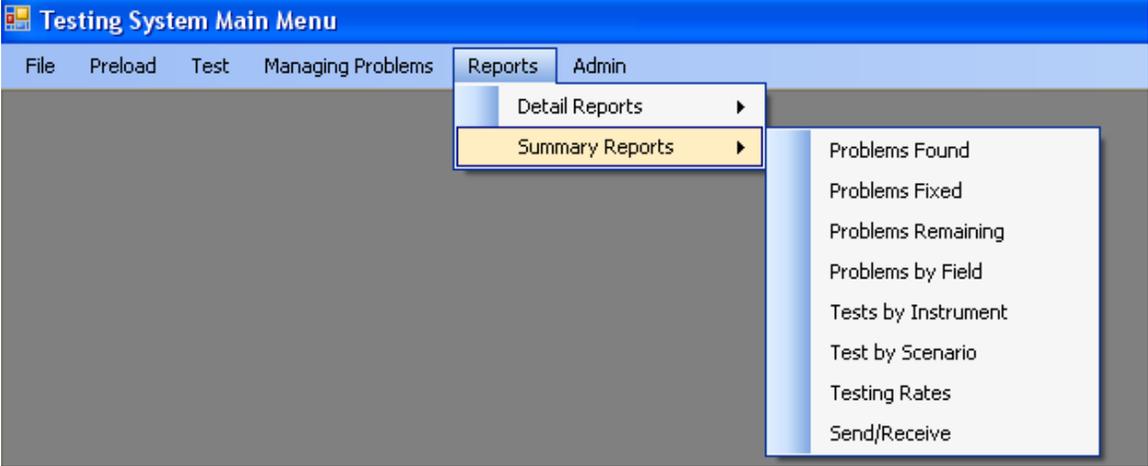


The report option allows the programmer to easily view the problems that have been reported. Previously, the programmer would have to pick out the problems from a large document that could get up to 30 or more pages in length.

The tester was not forgotten either. They have a report in the Details Report menu called “Ready to Test.” As the name indicates, this report lists all the items that the programmer has fixed and the tester needs to review.

Several summary reports are available that provide information on test performance metrics such as number of problems found, fixed, and remaining for each data model tested.

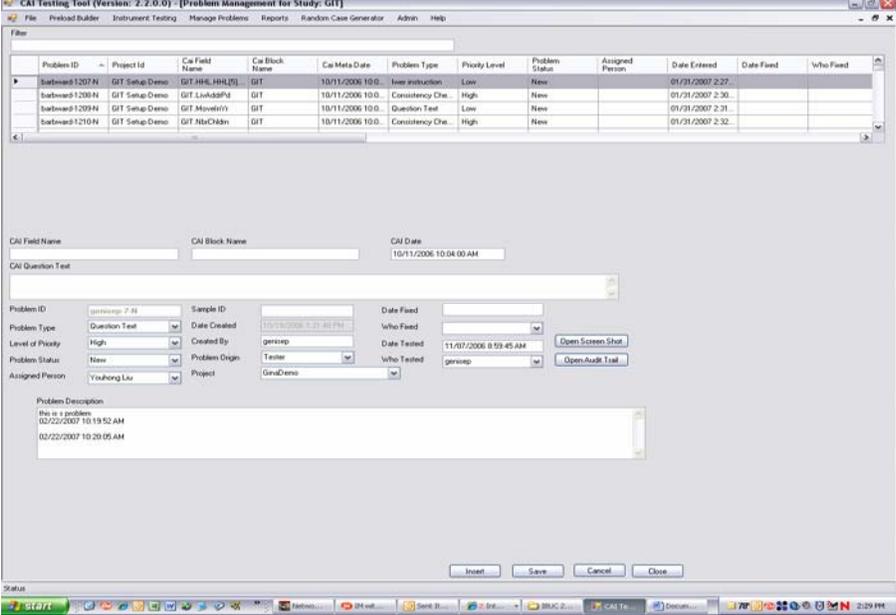
Figure 7. Summary Reports



3.4. Problem Management

The Manage Problems feature is used by project managers and testing coordinators to review and manage the bugs/enhancements that are reported during testing. For example, the testing coordinator can change the problem status or level of priority of a problem on the screen. The programmer uses the Manage Problems module to locate New and Not Fixed problems. When a problem is resolved and ready for re-testing, the programmer sets problem status to Ready to Test, and the Testing Note is made available to testers once again for re-testing.

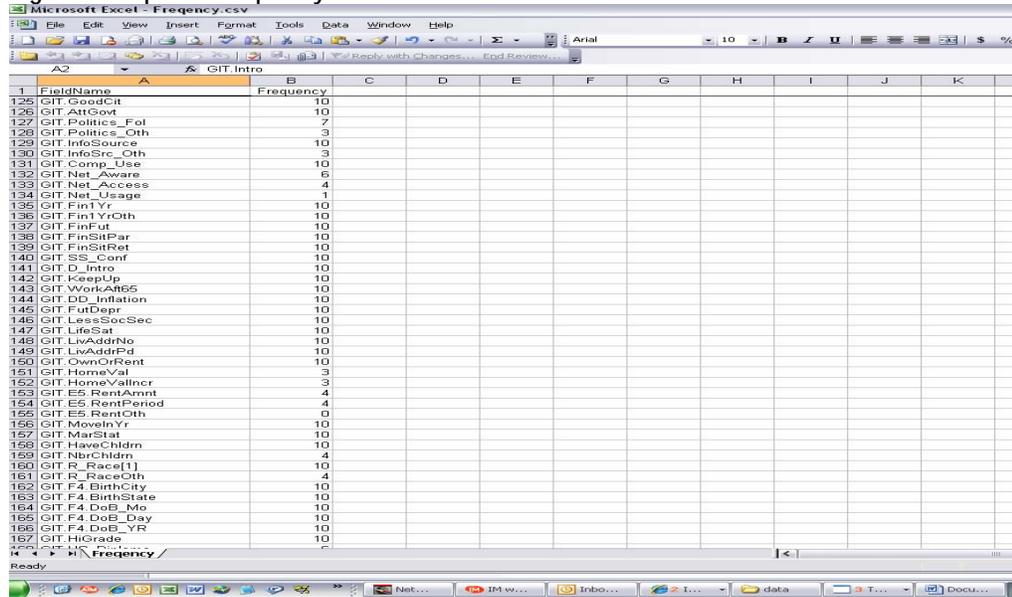
Figure 8. Manage Problems Screen



3.5. Random Case Generator

Automatically generating cases with random data in the Random Case Generator has the advantage of testing the entire instrument and every path within the instrument over many cases. It allows for focused generation of data and testing targeted skip patterns. The output generated, a file containing frequencies for all fields, is useful in determining whether or not all questions that should be on a path and answered actually have a response.

Figure 9. Output Developed by Random Case Generator

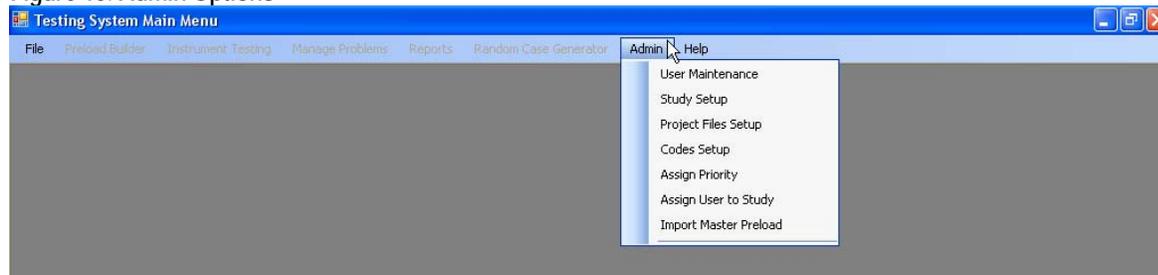


Field Name	Frequency
125 GIT_GoodCh	10
126 GIT_AttGovt	10
127 GIT_Politics_Fol	7
128 GIT_Politics_Oth	3
129 GIT_InfoSource	10
130 GIT_InfoSrc_Oth	3
131 GIT_Comp_Use	10
132 GIT_Net_Aware	6
133 GIT_Net_Access	4
134 GIT_Net_Usage	1
135 GIT_Fin1Yr	10
136 GIT_Fin1YrOth	10
137 GIT_FinFut	10
138 GIT_FinSHPar	10
139 GIT_FinSHRet	10
140 GIT_SS_Conf	10
141 GIT_D_Intro	10
142 GIT_KeepUp	10
143 GIT_WorkAtt65	10
144 GIT_DD_Inflation	10
145 GIT_FutDepr	10
146 GIT_LessSocSec	10
147 GIT_LifeSat	10
148 GIT_LivAddrNo	10
149 GIT_LivAddrPd	10
150 GIT_OwnCrRent	10
151 GIT_HomeVal	3
152 GIT_HomeValIncr	3
153 GIT_E5_RentAmnt	4
154 GIT_E5_RentPeriod	4
155 GIT_E5_RentOth	0
156 GIT_MoveInYr	10
157 GIT_MarStat	10
158 GIT_HaveChldrn	10
159 GIT_NbrChldrn	4
160 GIT_R_Race[1]	10
161 GIT_R_RaceOth	4
162 GIT_F4_BirthCity	10
163 GIT_F4_BirthState	10
164 GIT_F4_DoB_Mo	10
165 GIT_F4_DoB_Day	10
166 GIT_F4_DoB_YR	10
167 GIT_HiGrade	10

3.6. Administrative Features

CTT was designed to be a code driven system and contains an Admin module that allows CTT to be customized to meet different user needs without requiring changes from a programmer. You indicate in CTT the rights assigned to different users. This feature is only available for users with Admin rights. Some tasks completed in the module are adding users, setting up new studies, adding or modifying codes, assigning users to a study and importing preload files.

Figure 10. Admin Options



4. Summary

In summary, the CAI Testing Tool (CTT) is a very useful tool to manage the problem-reporting process for Blaise instrument testing. The system provides the testers an easy interface to record problems and produce reports to use during testing cycles. CTT provides the programmers with accurate details regarding where the problem is located and provides reports to help make the changes to the Blaise application.

During the last year of using CTT, we have been capturing end user's suggestions for improvements. These enhancements include:

- Adding the ability to copy all the Problem Type/Priority codes setup from one study to another one in the Admin menu item- Assign Priority. The studies are using the standard items and may change one or two. This enhancement would speed up the project setup and improve the usability of the tool.
- Enhancing CTT to be able to record problems/enhancements while testing in the Sample Management System SurveyTrak.
- Enhancing the Problem Management Window with several suggestions such as adding the ability to delete unwanted records; adding a fast way to change a group of problems to ready to test; adding the ability for the end user to create filters to control what displays for them in the Problem Mangemet window.
- Adding two new Detail reports: All Problems and Closed Problems. These reports would be useful to the testing coordinator to review all the problems regardless of status. Both reports are sorted by project, then Blaise field name.
- Adding the Synchronize Database feature to the CTT menu. Currently it is a shortcut on the desktop.

The addition of the above features or changes will make this system more robust and user friendly. We have found that the CTT has achieved what it set out to accomplish and that the system is greatly appreciated by the users.

Blaise Testing

Margaret Tang and Daniel Collison, Statistics Canada

1. Introduction

Statistics Canada (STC) has used Blaise to develop survey collection instruments since 1999. Currently, there are more than 200 Statistics Canada surveys in production. This includes monthly, quarterly, and annual surveys across the agricultural, business, and social areas for head office and the six regional offices across Canada.

In Statistics Canada, each survey typically uses the Blaise build¹ that was chosen at the start of the cycle. Once the survey is in production, the development managers are reluctant to change to another version of Blaise for future cycles. Survey managers only change to a new Blaise build if there are new features/fixes required for the next production cycle.

Different surveys have different components and utilize different features of the Blaise system. To minimize testing efforts and to lower the risk of discovering costly issues in later stages, it is necessary to establish a standard test strategy for the core features of the Blaise system used by most Statistics Canada surveys.

2. Blaise Testing Complexity

A Blaise survey instrument includes multiple questions and complex routing rules. Each survey instrument can have user defined elements such as data types, edits/checks for question fields, outcome codes and multiple interviewing languages. Moreover, different Blaise surveys can also have customizable features such as menu items to interact with external applications or utilize external databases. When all of these factors are taken into account, the Blaise testing complexity increases significantly.

Due to time constraints, Blaise survey testing concentrates primarily on the Data Entry Program for survey content and behavior. However, the Blaise system consists of many components including survey development, batch processing, data entry, call scheduling, data storage and maintenance (e.g. Hospital and DayBatch). Different versions of Blaise introduce new and modified features in different components that may not always work with existing survey instruments. Problems may arise with the Call Scheduler or Hospital as they are not part of the Blaise survey testing focus. To handle the complexity of testing these Blaise components, it becomes essential to establish a testing process to validate a Blaise build before recommending it for development.

3. Blaise Build Testing Goals

Statistics Canada has chosen to take advantage of the release of Blaise 4.8 to establish a standardized and reusable test process that systematically evaluates a new Blaise build. This will be accomplished by assessing the compatibility of the new build against current production needs, validating its ability to co-exist with other builds, and determining the changes required within a survey for it to be able to adopt the new Blaise build. The

¹ Blaise build – a specific version or release of the Blaise survey processing system

second goal is to establish a set of acceptance criteria that a Blaise build must satisfy before it can be recommended for survey development use. The third goal is to provide an estimate for time and effort needed to migrate to the new build. And, the final goal is to investigate the new features of a build and make recommendations on their applicability for STC surveys.

4. Blaise Build Testing Process

The Blaise build testing process has been established using the Rational Unified Process approach. This process considers the test effort from the perspectives of roles, activities, and artefacts.

4.1 Blaise Build Testing Roles

4.1.1 Test Manager

This role leads the overall test effort. This includes quality and test advocacy, resource planning and management, and resolution of issues that impede the test effort.

4.1.2 Test Designer

This role defines the test approach and ensures its successful implementation. This includes identifying the appropriate techniques, tools and guidelines to implement the required tests, and providing guidance on the corresponding test resources requirements.

4.1.3 Test Analyst

This role identifies and defines the required tests, monitors detailed testing progress and results in each test cycle, and evaluates the overall quality.

4.1.4 Tester

This role conducts tests and logs the outcomes of his/her testing.

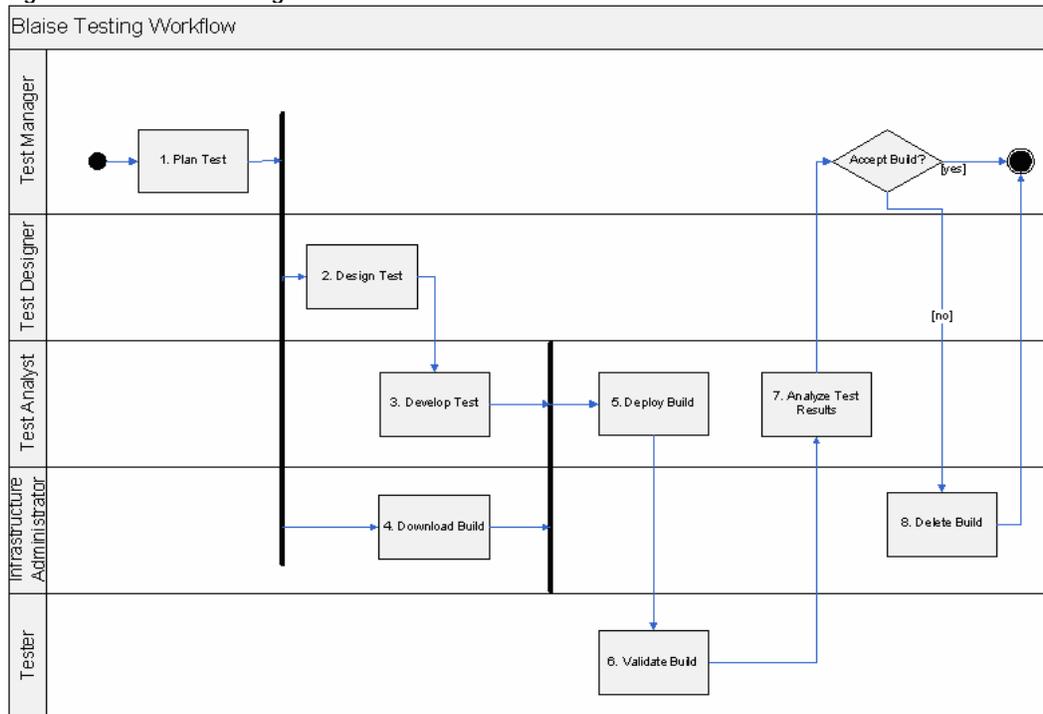
4.2 Blaise Build Testing Activities

The objective of the Blaise Testing Workflow is the verification that the Blaise under review will satisfy the common usage scenario and the acceptance criteria for Blaise applications in Statistics Canada. This involves the following steps:

- Establishing the scope and focus of the test
- Designing and implementing the required tests
- Performing the tests and reporting the problems discovered

Different testing roles are responsible for different activities in the workflow. As time progress, manual tests will be gradually enhanced or replaced by automated tools to improve high testing quality and/or lower testing cost.

Figure 1. Blaise 4.8 Testing Workflow



4.3 Blaise Build Testing Artefacts

4.3.1 Master Test Plan

This artefact defines the goals and scope of the test project, the items being targeted, the approach to be taken, the resources required and the deliverables to be produced.

4.3.2 Test Case

This artefact defines a set of test inputs, execution conditions, and expected results for the purpose of making an evaluation of some particular aspect of a target test item.

4.3.3 Test Bed

This is an environment created for Blaise build testing purposes. It includes commonly used Blaise and non-Blaise applications in STC.

4.3.4 Test Script and Procedure

This artefact defines steps to be executed automatically (script) or manually (procedure).

4.3.5 Test Log

This artefact provides a detailed status of the test effort and notes.

4.3.6 Test Evaluation Summary

This artefact presents a summary and analysis of the test results for review and assessment. It may also give recommendations on how to adapt the Blaise build.

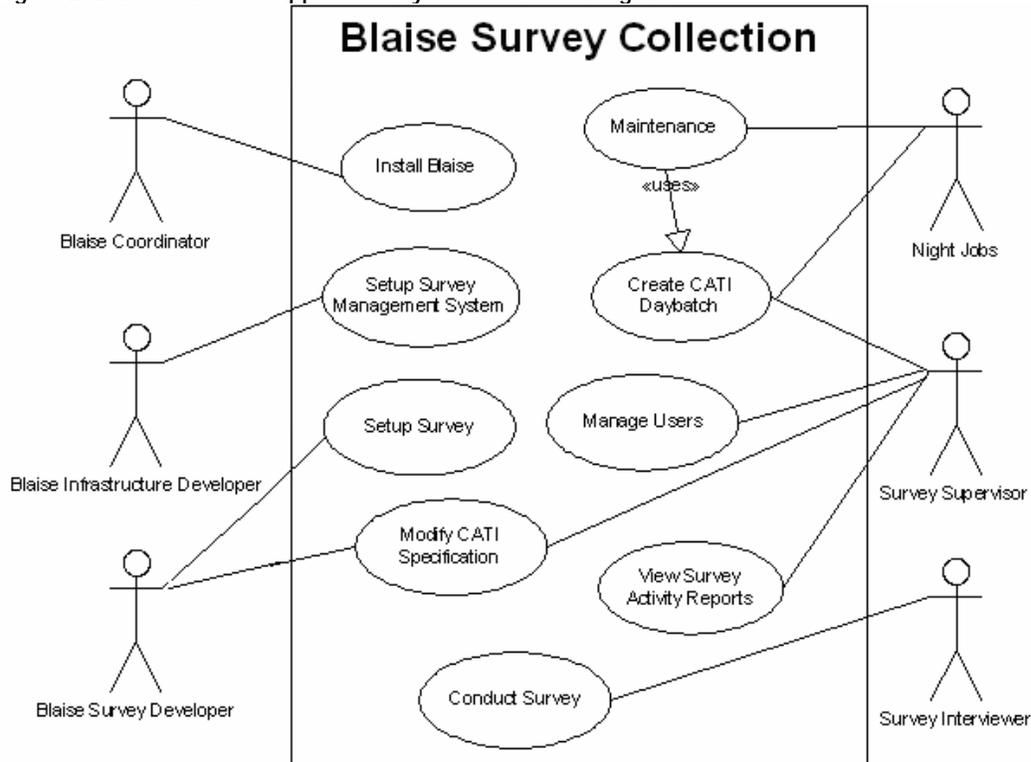
Table 1. Summary of Test Activities and Artefacts

	Activities	Artefacts
Test Manager	Plan Test	Master Test Plan
Test Designer	Design Test	Test Cases, Test Bed
Test Analyst	Develop Test	Test Scripts/Procedures
	Analyze Test Results	Test Evaluation Summary
Tester	Execute Test	Test Logs

5. Blaise Build Testing Approach

The Test Manager first identifies the scope and focus of the test in the Master Test Plan.

Figure 2. Blaise Collection Application System Use Case Diagram



The Test Designer then creates a suite of Test Cases and a Test Bed that will cover the scope. For example, for the Install Blaise Use Case, there will be one Test Case for installing the Blaise core system and another Test Case for installing the Blaise services.

One Use Case can generate multiple Test Cases based on functionality or options. Test Cases can also cover other aspects such as usability, reliability, performance and supportability (including compatibility and existing infrastructure constraints)

Based on the Test Cases, the Test Analyst will create Test Scripts or detailed Test Procedures for the Tester. For example, to simulate survey case load, Blaise Emulator scripts will be used to establish multiple Emulator sessions.

Next, the Tester will execute Test Scripts and record the results in the Test Logs so the Test Analyst can create the Test Evaluation Summary for a Blaise build. Finally, the Test Manager will accept or fail the build based on the Test Evaluation Summary.

The cycle restarts with the Test Analyst modifying Test Scripts/Procedures to reflect the peculiarities of the next build that is received. If the test scope is changed, the Test Designer will create new Test Cases and the Test Analyst will create the necessary Test Scripts/Procedures for the Tester.

6. Blaise 4.8 Build Testing Experience

There are 3 main focus areas in the Blaise 4.8 Build Testing:

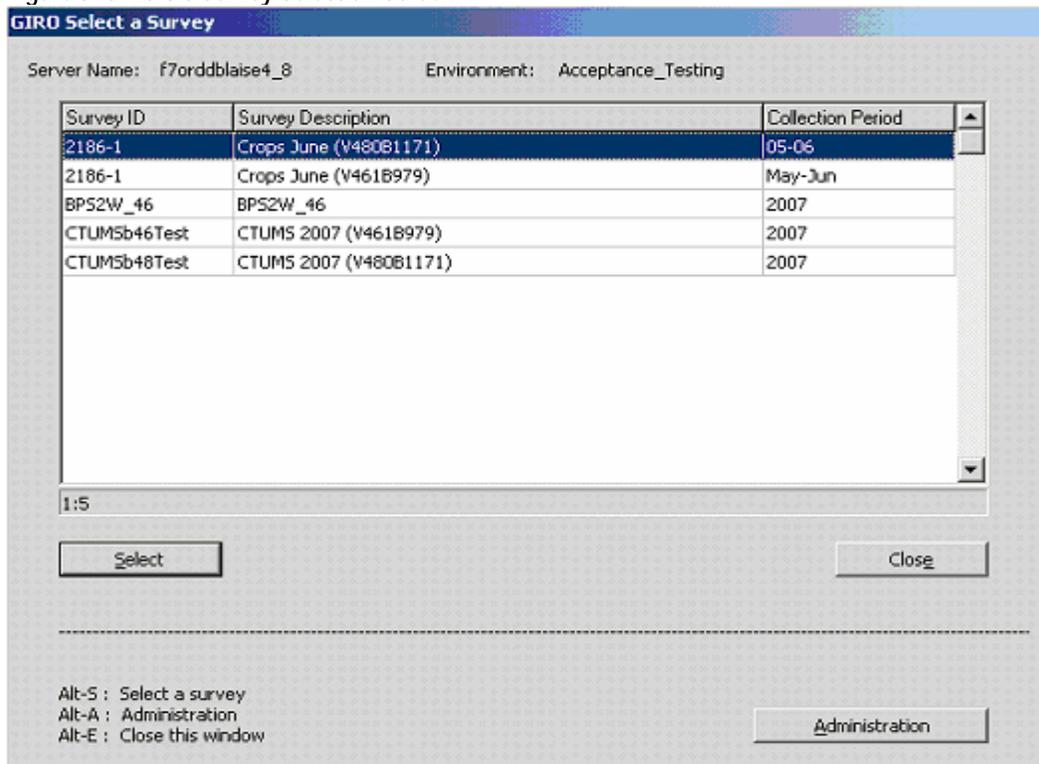
- Blaise Server Architecture: Blaise Database Service and Blaise CATI Service
- Backward Compatibility: The support of common survey collection activities
- New Features: Blaise DataLink, Blaise CATI events and Maniplus Interchange

6.1 Blaise 4.8 Build Testing Test Bed

The Blaise 4.8 Build Testing Test Bed consists of a Survey Management System, a Test Survey and several Survey Collection Applications.

The Survey Management System used is GIRO 3.0 (General Interface for Regional Offices), developed using Blaise 4.6 (build 979). It handles Survey Management and User Management functionalities for surveys in Statistics Canada's regional offices.

Figure 3. GIRO 3.0 Survey Selection Screen



The Test Survey used is CTUMS 2007 (Canadian Tobacco Use Monitoring Survey). It is developed using Blaise 4.6 (build 979). A 4.6 copy is used as the benchmark and its source code is recompiled in 4.8 and setup for Blaise 4.8 Testing.

The Survey Collection Applications included in the Test Bed are VB Browser (Survey Data Browser) and CATI Specification Regional Interface. They are both Visual Basic applications that interact with Blaise using the BCP (Blaise Component Pack).

Survey Activity Reports are developed using SAS 9.1 with Excel. Manipula scripts and Delphi DLLs used for Data Extraction complete the Test Bed.

6.2 Hardware Resources

Table 2. Summary of System Resources

	Operating System	Comments
Blaise Testing Server	Windows 2003	1 Virtual Server
Development Server	Windows XP	1 PC used for configuration test
Developer PC	Windows XP	2 developer PCs
Client Testing PC	Windows XP	10 PC from Testing Lab (as needed)

6.3 Blaise 4.8 Server Architecture

When testing the Blaise 4.8 Services, the Blaise 4.8 Test Survey is used for Data Entry and the Blaise Emulator is used to simulate a multi-user working environment. Memory and CPU usage of the server are monitored using Spotlight for Windows. Network Traffic is monitored by STC's Infrastructure Management team.

Table 3. Summary of Test Status

Version	Status	Notes
4.80.1129	Failed	Slow Blaise Database Service Performance for Batch Jobs
4.80.1146	Failed	Data Entry Program unable to work with CATI Service
4.80.1159	Failed	Slow/Unstable CATI Specification when modifying large file Slow WAN performance when all items are in Head Office

6.4 Backward Compatibility

Blaise 4.8 is deployed on the Blaise Testing Server in the same fashion as Blaise 4.6. The Test Bed is used to evaluate the common features used in STC. It also investigates the steps needed to upgrade to Blaise 4.8.

Table 4. Test Items and Blaise Features Covered

Test Item	Function	Blaise Feature
GIRO	User Manager – Update CATI	Blaise CATI Specification Program
	Select/Browse	Blaise API (VB 6 Application)
	CATI Specification	Blaise CATI API (VB 6 Application)
	Reports	Interaction with SAS and Excel
CTUMS2007	Compile	Blaise Parser (Command Line)
	Load	Manipula
	Functions Menu	Maniplus
Maintenance	Interview	Data Entry Program
	Check Data Storage Integrity	Blaise Hospital Program
	Data Extraction	Manipula
	Day Batch Creation	Blaise CATI Management Program
	BTH Extraction	Blaise Transaction History files
	Report Generation	Manipula and SAS
	Audit Trail Extraction	Audit Trail (Blaise Audit.dll)

Table 5. Summary of Test Status

Version	Status	Notes
4.80.1171	Suspended	Suspended due to news of API changes in build 1180
4.80.1180	Suspended	Suspended due to availability of build 1190
4.80.1190		First Production Blaise 4.8 Build

Table 6. Summary of Common Issues

Component	Issue	Notes
Blaise Parser	Parser Rules Tightened Up	Code is checked more rigorously
CATI Specification	New Format (.btr)	Existing .bts will need to be migrated to the new format
CATI Management	Microsoft XML 4.0 required	All users will need to install the Microsoft XML 4.0 locally

6.5 Other Blaise Features

Individual Test Scripts are created for Blaise DataLink, Blaise CATI Events, Alien procedures and Maniplus Interchange. These are used to evaluate possible future survey development and potential deployment/maintenance issues in user environments.

7. Conclusion

Blaise 4.8 Build Testing is a long process. Establishing a standard test suite helps reduce testing effort and improves testing quality. It also facilitates early error detection and problem reporting. As time goes on, more features will be added to the standard test suite. With automation, more extensive testing can be done in much less time. Eventually, the cost and risk of upgrading to a more recent Blaise build will be much lower and will encourage survey development managers to explore the full potential of Blaise.

8. References

IBM Corporation, Rational Unified Process Version 7.0, 2005

Case Scenario Library for Testing Large Blaise Applications

Rhonda Ash and Jason Ostergren, The University of Michigan

1. Introduction

In the last couple of decades, the use of computers for conducting interviews has allowed an increasing amount of complexity and customization to be achieved in the mechanics of a survey instrument without necessitating a corresponding increase in the burden on the interviewer or respondent. Because of this trend, it is now possible to design legitimate specifications that nonetheless overwhelm our ability to develop reliable survey instruments using the presently available tools. The Health and Retirement Study (HRS) at The University of Michigan has such a design. In 2004, the average HRS respondent was asked 425 questions out of a possible 11,745 over the course of 87 minutes. These figures were the result of approximately 76,000 conditional statements and 220,000 active lines of code in the logic of the instrument. Variation did not simply extend to the logic, however. HRS made use of what may be an unprecedented number of “fills” which had the potential to customize each of these questions to take into account a variety of factors such as extensive preloaded data that HRS uses, or proxy wording, or deceased respondents. All of this in a survey development environment (Blaise) which includes no debugger and scarce resources with which to test in any but the most time consuming manual fashion. In order to cope with the contest between complexity of design and reliability in programming, HRS developed a number of tools designed to improve debugging and testing in a Blaise environment with the goal of achieving an appropriate level of reliability. This paper will trace the evolution of this set of tools and then dwell in detail on the latest, a library of “scenarios” that combines and builds on them.

2. HRS Overview

The HRS is a complex longitudinal survey covering a wide range of health issues and economic concerns related to aging and retirement. The average interview lasts more than an hour and is administered to over 22,000 respondents. The HRS began in 1992 and had been carried out for three waves at two-year intervals before it was merged in 1998 with its sister study AHEAD, which covered an older cohort and a somewhat different content area. At that time two new age cohorts were added, and a “steady-state” design was born which would avoid eventual obsolescence by bringing in new cohorts at regular intervals (another was added in 2004). That change dramatically increased the capacity of HRS to address the effects of events and policy changes within its scope. It also allows HRS to aspire to a very long lifespan with continual expansion and adaptation of its content as these changes occur.

The complexity of the design is compounded by the need for multiple language types, not only for Spanish interviews, but also for the special wording needed for proxy interviews. HRS conducts proxy interviews with living respondents when necessary and also attempts up to two exit interviews in waves following the death of a respondent. In 2002, the exit interview, which had previously been programmed as a separate instrument, was merged into the main instrument with a resulting increase in scheduling efficiency at the expense of yet more complexity in the code.

Further, the code for the HRS instrument is rather unwieldy due to a host of other factors. For example, HRS makes use of a large amount of preloaded data (as many as 400 variables) particularly for family members and co-resident people who may be helping the respondent or influencing the respondent's life financially. Most importantly, the spouse or partner of the respondent, although interviewed separately, nonetheless has much of their information loaded for confirmation purposes. These data are fed into a series of tables at the start of the interview. This poses a particular problem because no other content can be reached and thus tested without first passing through a series of elaborate tables and sequences based on this preload. The main content of the interview is divided into approximately two dozen sections, each of which need to be programmed separately. This is because we rely on different sets of analysts and researchers to determine what additions, deletions, and changes will be implemented in each section for each new wave. Changes typically involve alterations to question text or to the 7,000+ fills in order to clarify or customize the language. Additionally, substantial changes to the logic of these sections are frequently requested in order to add new content, save interview time by targeting questions better, or accomplish any number of other goals.

3. HRS Testing Tools

At this point, a sense of the scope of the problem of ensuring the reliability of the HRS instrument should be emerging. In 2001, the study was in the middle of a transition from SurveyCraft to Blaise as the survey programming language. What had been a well-established set of code had to be set aside along with all of the tools that the staff had become accustomed to for checking the integrity of the instrument. Though Blaise is a conventional-looking language, with moderately customizable tools and a decent application programming interface (API), it had shortcomings, particularly for a survey the size of HRS, which became apparent during that transition year.

Many of these early difficulties tended to result in greater complexity. Performance problems are a good example of this. Surveys ran very slowly at first because large amounts of data derived from preload had to be accessible to different parts of the instrument and generated parameters proliferated as a result. This played havoc with the selective checking mechanism and caused the time gap between questions to become unacceptably long. Eventually these problems were solved as project staff became more familiar with Blaise, but the solutions, such as adding copies of arrays or requiring certain sections to be locked out after completion, added further to the complexity of the instrument. And as complexity increased, so did the difficulty of testing the instrument.

Off the shelf, Blaise lacks the sort of debugging and testing capabilities and systems that are needed for a complex survey like HRS. Foremost among the testing needs of HRS was some way to run an interview with real or simulated preload. Without this capability, it was nearly impossible to determine whether the new Blaise code being written would work in a real-world environment, especially given the variety of different interview patterns which are determined by preload. Since Blaise uses a proprietary database technology that was (and remains) difficult for data processing staff without extensive programming knowledge to access, some sort of case management system was needed that could read preload in a standard database format and then load it into the Blaise database, at the same time performing a series of manipulations required by the HRS design. A program called SurveyTrak, developed for field use by The University of Michigan, proved too cumbersome to use during development because the design precluded the study staff from updating preload or Blaise code with a quick turnaround.

In the end, HRS began to develop its own case management programs which allowed for easy manipulation of preload and quick release of code fixes for testing. Without this tool, it is difficult to imagine the 2002 wave of HRS having made it into the field in a timely manner without a disastrous number of errors and bugs. Also, without this basic capability, some of the other important tools that will be discussed here would not have been possible.

As background for the Scenario Library tools that are the subject of this paper, what follows is an explanation of the development of HRS testing applications, particularly in the wake of the 2001-2002 development period. Once that first Blaise version of the survey was in the field, and a fair number of subsequent updates had been issued, we assessed the effectiveness of our programming and testing practices and determined that there were a number of areas where considerable improvement could be realized by the development of additional testing applications. There were two main issues that needed to be addressed at that time: the quality of error reporting and the ability to reproduce reported errors.

The solution to the reporting problem was the simpler of the two. Already in 2001, HRS had an error reporting database where problems were collected. However, the quality of these reports was decidedly mixed because it was completely manual. It was often unclear where in the instrument an error had appeared and what circumstances had caused it to appear. As a result, programmers and testers typically expended enormous amounts of time repeatedly attempting to key their way to the point in the instrument where the problem was alleged to have occurred. Oftentimes, things such as unreported alterations to preload by the reporter conspired to make it very time-consuming to find the reported error. Vague error reports are certainly a common problem, but in survey programming they should be far less burdensome than in other areas of software development. Blaise is a narrowly focused scripting language that is solely geared towards surveys, and the programmer and user of a Blaise instrument operate in a very circumscribed environment. It therefore ought to be quite easy to identify and reproduce the actions which produce an erroneous behavior. The first step toward this goal was to automate the reporting process. Using the Blaise API and the menu functionality of the Blaise data entry program (DEP), HRS produced a tool using Visual Basic 6 that would collect metadata directly from the DEP and send it to the bug database along with the audit trail file from the interview in progress. This would occur when the user selected an option that we added to the DEP menu, and at the same time a window would appear to prompt the reporter for a description as well.

This brings us to what has become the quintessential HRS testing tool which was the subject of an HRS paper at the 9th International Blaise User's Conference: a keystroke replayer. The Survey Research Center at The University of Michigan has been very interested in audit trails for some time from both a research perspective: what can be learned from "paradata," and from a quality assurance perspective: that is, audit trails can be used to monitor interviewer performance. HRS hoped to use them for another form of quality assurance: testing and debugging. The idea was to collect audit trails from error reports, as described above, or from other testing activities. A separate application would then attempt to reproduce the interview that generated the audit trail when a programmer had need of it, and possibly again later for a tester to verify that a fix, for example, was working. What is more, it was hoped that this would aid in other testing needs, such as language testing, by allowing certain cases to be replayed under different conditions. This keystroke replayer was developed in Visual Basic 6, and turned out to be the most

elaborate of all the HRS tools. In order to function, it required a number of separate programs including the DEP, all running in concert and maintaining communication and synchronization through Windows API calls as described in our previous conference paper. This program was completed in 2002, and did indeed produce a noticeable improvement in the quality and efficiency of debugging and testing during subsequent development periods.

However, there were a couple of problems with our otherwise successful keystroke replayer. First, new versions of Blaise from time to time bring with them small changes to the DEP that require reworking of the program. Second, due to the complicated nature of what we were doing, it proved difficult for some of the less technical users to become comfortable with the replayer. As a result, in late 2004, HRS embarked on a project to entirely rewrite the suite of testing tools and add new automated testing features.

One part of this process was to build a new case manager. The new version was written in a .NET language (C#) in order to take advantage of the .NET libraries and its better handling of objects. The most important feature of this rewrite is that it can easily switch among a variety of preload data storage formats including SQL Server, MS Access, and XML. Our practice now is to store preload in a common permanent SQL Server database in which new records (often modified copies of old ones) are allowed, but the updating or deleting of previously existing records is not. Temporary changes to preload are allowed when dumped to XML, because such files can be preserved for later reference. What this has allowed us to do is to track every preload change made by a user, thus eliminating a major source of variability in reproducing problems. Now, for example, with each error report, a copy of preload in XML format is stored along with the audit trail in our SQL Server bug database. This has actually had a dramatic effect on our testing efforts because testers have become less constrained when altering preload, allowing them to more fully evaluate the ability of the instrument to cope with different sets of preload.

The most important new features of the testing system were built in answer to the user-friendliness problems of the keystroke replayer. Since the DEP was not easy to work with from a programming perspective, HRS pursued two new strategies to supplement the replayer. They both rely on the fact that the Blaise API exposes the functions of the DEP, so it is possible to simulate its operation and remove the actual DEP from the equation altogether. First, we added an option to simulate a replay in the background, which has proved very popular with users due to its ease of operation. Second, a rudimentary DEP was built in VB.NET that used the API and displayed a user interface that could be manipulated at will. Both of these solutions eliminated the need to “send” keystrokes and query the DEP to maintain synchronization. As usual, there were problems despite the good reception of the new tools. We discovered that in certain obscure situations, the Blaise API does not operate precisely the same as the DEP. Statistics Netherlands was notified and they confirmed that we had discovered a new bug. A future resolution of this issue, resulting in a high degree of confidence that the behavior of the API matches that of the DEP, could even lead to the development of a full-fledged custom-built DEP. Despite these minor issues, the new tools have yielded even better results in 2006 than were obtained in prior waves. Overall, in the absence of built-in debugging features in Blaise, these tools have allowed HRS to release a complex survey instrument every two years with reasonable confidence in its reliability.

4. Scenario Library

Prior to the 2006 development period, work began on another testing tool to address the long-held desire for a way to handle HRS “scenarios” in an automated way. In HRS parlance, scenarios are constellations of preload data and response patterns that correspond to a particular category of respondents. For many years, these were developed by reading the specifications and hand-entering combinations of preload and response values into spreadsheets describing each significant category of respondents in each section. A tester was then required to interpret these scenarios and enter the values into the instrument in order to verify its accuracy. In combination with many of the tools and techniques described in the section above, HRS has developed a new set of tools to automate this process. This is described in detail below.

5. Scenario utilities

5.1 Overview and Purpose

With a multi-leveled interviewing application, including language controls, subject content, multiple interview modes, and re-interview constraints, it is very important to be able to test each dimension. Keystroke files provided by the Blaise process allow us to validate the application in a systematic manner. HRS has created a mechanism to store those files in a library that can recall them at a later date, alter the original test, and quickly guarantee instrument accuracy.

HRS was able to reduce error difficulties by conceiving of and developing testing utilities that combine into a system called the HRS Case Manager (HRS-CM). Together, these programs allow testers to make use of stored keystroke files that are produced as a by-product of Blaise test interviews. These files are categorized and stored for reapplication.

To ensure maximum coverage and reduce the time needed for testing, the keystroke files for specific scenario paths are given identifying information such as a scenario number, mode of interview, section of test focus, and several major variables that control flow. HRS commonly refers to the course that an interview follows through an instrument as a “path” or as the “flow.”

5.2 Implementation

There are three components to the testing process, each requiring different ways of handling applications: 1) preload – information that we have collected in the past about a respondent’s finances, family or medical history that needs to be brought forward into this wave’s data and updated, 2) a Blaise datamodel – the program specifications that allow the collection of data, and 3) a scenario - the combination of preload, path, and content though the instrument that is later used for verification of accuracy. We will detail each component and describe the mechanism designed to incorporate it into a coherent system.

5.3 Preload process lifecycle

Preload can determine the path taken through the application. As an example, consider a respondent we have talked to before from whom we have collected baseline information

versus a respondent that we have never interviewed. These two respondents would take different, but overlapping paths through the application. We would not need to ask the baseline questions of the re-interview respondent, but we would need to ask them of the new respondent.

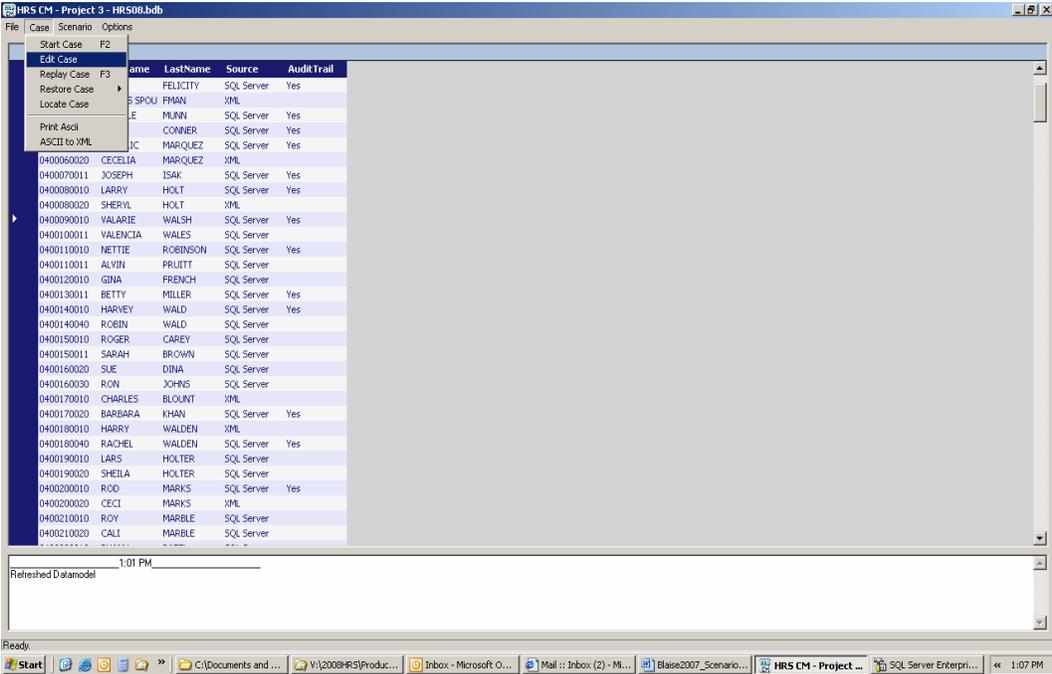
There are several major flow control fields that greatly influence the flow of the HRS application. These were considered when a preload library was developed. A set of common preload values were created from real data. These cases were then altered and any identifying data was changed. This set, which numbered around one hundred cases, became the basis for the HRS_CM scenarios. As the testing cycle advances, the testers have made additions to the preload library for specific testing needs.

These data are currently stored in several related tables on an SQL server, and accessed through a series of programs that allow update and retrieval for usage. Below we provide an example of a preload table with each row representing part of a case. Each case includes data for most of the 400+ variables.

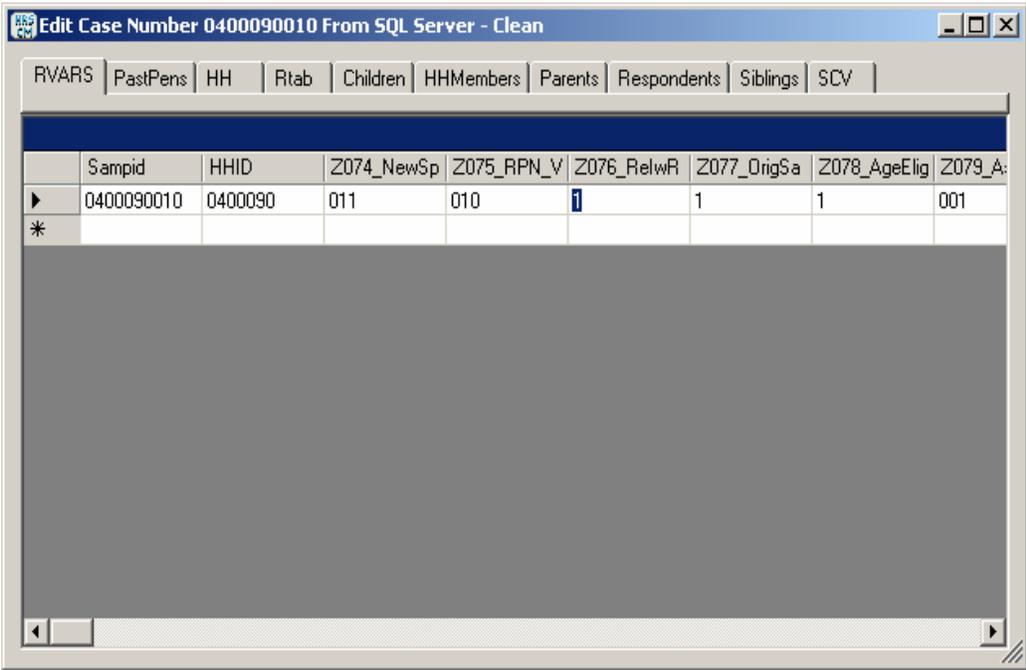
Sarnpid	HHID	2074_NewSpSeed	2075_RPN_V	2076_RetwR_V	2077_OrigSamp_V	2078_AgeElig_V	2079_AskDOB_V	2080_MarStat_V	2081_SpDie_V	2082_WaveId_V	2083_UnfMedCost	2084_UnfNHosp	2085_L
0401130010	0401130	011	010	1	1	1	001	1	5	0400810010	3	1	1
0401130040	0401130	041	040	1	1	1	001	1	5	0400810040	2	2	2
0401140010	0401140	011	010	5	1	1	001	4	5	0400900010	2	1	3
0401150010	0401150	012	010	1	1	1	110	1	5	0400620010	3	1	3
0401150011	0401150	<NULL>	011	5	5	1	111	1	5	0400620011	2	2	2
0401160010	0401160	011	010	1	1	1	001	1	5	0400540010	2	3	1
0401160020	0401160	021	020	1	1	1	001	1	5	0400540020	2	2	2
0401170010	0401170	011	010	1	1	<NULL>	001	<NULL>	5	0400400010	3	3	2
0401170020	0401170	021	020	1	1	<NULL>	001	<NULL>	5	0400400020	1	3	3
0401180010	0401180	011	010	1	1	<NULL>	001	<NULL>	5	0401170010	3	3	2
0401180020	0401180	021	020	1	1	<NULL>	001	<NULL>	5	0401170020	1	3	3
0401190011	0401190	012	011	1	5	1	001	5	5	0400100011	3	3	1
0401200010	0401200	011	010	1	1	1	001	5	5	0401200010	3	3	1
0401210010	0401210	011	010	1	1	1	001	5	5	0401200010	3	3	1
0401220010	0401220	011	010	1	1	1	001	3	5	0400970010	1	3	3
0401230010	0401230	011	010	1	1	1	001	5	5	0400200010	3	3	1
0401240010	0401240	011	010	5	1	1	001	2	5	0400060010	3	3	2
0401240020	0401240	021	020	1	1	1	001	3	5	0400060020	1	3	3
0401250010	0401250	011	010	5	1	1	001	2	5	0400060010	3	3	2
0401250020	0401250	021	020	1	1	1	001	3	5	0400060020	1	3	3
0401260010	0401260	011	010	5	1	1	001	2	5	0400060010	3	3	2
0401260020	0401260	021	020	1	1	1	001	3	5	0400060020	1	3	3
0401270010	0401270	011	010	5	1	1	001	2	5	0400060010	3	3	2
0401270020	0401270	021	020	1	1	1	001	3	5	0400060020	1	3	3
0401280010	0401280	011	010	1	1	1	001	4	5	0400030010	2	1	3
0401290010	0401290	011	010	5	1	1	001	4	5	0400010010	2	1	3
0401300010	0401300	011	010	1	1	1	001	1	5	0400140010	3	1	1
0401300040	0401300	041	040	1	1	1	001	1	5	0400140040	3	3	3
0401310010	0401310	011	010	1	1	1	001	4	5	0400010010	2	1	3
0401320010	0401320	011	010	1	1	1	001	3	5	0400050010	1	3	3
0401330010	0401330	012	010	1	1	1	001	4	5	0400280010	2	2	1
0401330011	0401330	<NULL>	011	1	5	5	001	<NULL>	5	0400280011	1	1	1
0401340010	0401340	011	010	1	1	1	001	1	5	0400220010	2	3	1
0401340020	0401340	021	020	1	1	1	001	1	5	0400220020	2	2	2
0401350010	0401350	011	010	1	1	<NULL>	001	<NULL>	5	0400080010	3	3	2
0401350020	0401350	021	020	1	1	<NULL>	001	<NULL>	5	0400080020	1	3	3
0401360020	0401360	021	020	1	1	1	001	<NULL>	5	0400230020	3	3	3
0401360030	0401360	031	030	1	1	1	001	1	5	0400230030	1	1	1
0401370010	0401370	011	010	1	1	1	001	2	5	0400060010	3	3	2
0401370020	0401370	021	020	1	1	1	001	3	5	0400060020	1	3	3
0401380010	0401380	011	010	1	1	1	001	3	5	0400050010	1	3	3
0401390010	0401390	011	010	1	1	1	001	5	5	0400220010	3	3	1
0401400010	0401400	011	010	1	1	1	001	1	5	0400220010	2	3	1
0401400020	0401400	021	020	1	1	1	001	1	5	0400220020	2	2	2

When a case is called into use these data are copied into eight tables that have matching structure in the instrument prepared for this preload. This structure is ready to receive data.

Below is a sample of the available preload records (or cases) from the SQL server that has been saved in the HRS_CM system. This is the main screen from which the various testing activities can be activated.



By selecting a case from the HRS_CM system and choosing EDIT from the menu options, a tester is given the ability to view the preload values and alter data to perform a specific test as shown here.



If a tester has altered the preload for a case and will need that preload setup for future testing, they have the ability to save it as a new record and add it to the permanent preload list. If they have a specific need that would never have a broader applicability, they can choose to save it only to their testing folder on the network instead of saving it permanently to the SQL Server. A dialog box will appear asking them for their choice.



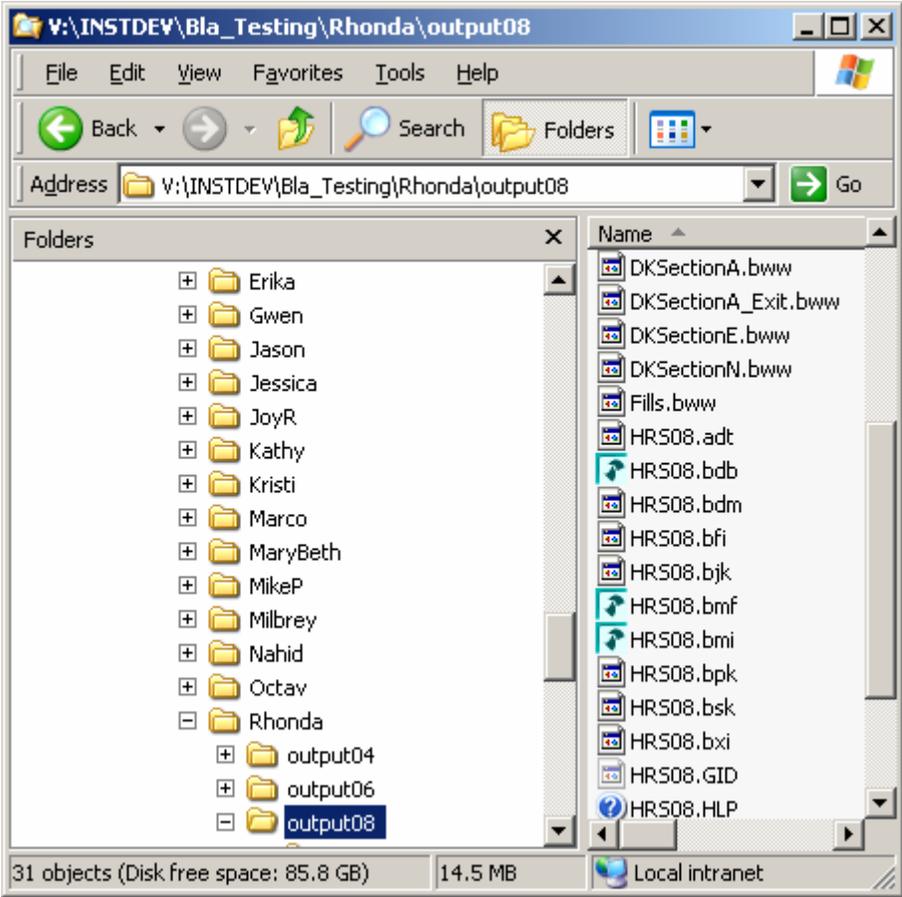
The temporary storage is done through the use of an XML file produced by this system (see below). This file contains changed and unchanged preload values.

```
<RVARs>
  <Sampid>0400430010</Sampid>
  <HHID>0400430</HHID>
    <Z074_NewSpSeed_V>012</Z074_NewSpSeed_V>
    <Z075_RPN_V>010</Z075_RPN_V>
    <Z076_RelwR_V>1</Z076_RelwR_V>
    <Z077_OrigSamp_V>1</Z077_OrigSamp_V>
    <Z078_AgeElig_V>1</Z078_AgeElig_V>
    <Z079_AskDOB_V>001</Z079_AskDOB_V>
    <Z080_MarStat_V>4</Z080_MarStat_V>
    <Z081_SpPDie_V>5</Z081_SpPDie_V>
    <Z082_Waveld_V>0055120010</Z082_Waveld_V>
    <Z083_UnfMedCost_V>2</Z083_UnfMedCost_V>
    <Z084_UnfNHHosp_V>2</Z084_UnfNHHosp_V>
    <Z085_UnfOthMedCos_V>1</Z085_UnfOthMedCos_V>
    <Z086_UnfPrescript_V>3</Z086_UnfPrescript_V>
```

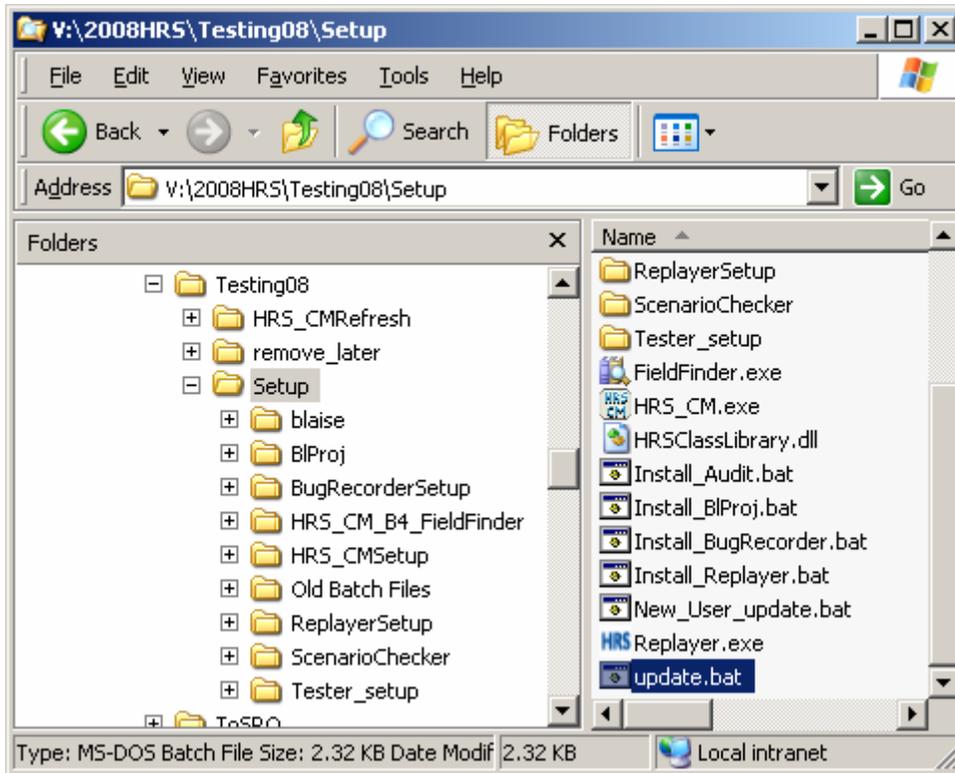
5.4 Datamodel process lifecycle

As the development phase progresses, changes and corrections are made to the Blaise source code. The programmer re-compiles and tests, then distributes the datamodel update for more extensive checking by testers. HRS has several programmers working at the same time in the application. Because of this the datamodels are time-stamped. This allows the testers to take note of when the datamodel was last updated and know when an issue is ready for re-testing.

HRS_CM is installed in a series of folder structures on a network. This structure allows each tester to have their own space, within which they can create their own testing environment without affecting other testers. This space includes saved watch-window settings associated with the tester's content area allowing quick confirmation of field content while testing, previous temporary preload files and played scenarios, along with the latest datamodel and instrument HELP file.



With a series of batch files that help move files from one location to another, we have been able to routinize the file updating processes. Below is an illustration of the folders used to set up a new tester with all needed files and a batch file that places files in the appropriate place and runs any installation application to activate programs.



Programmers also release new datamodels through the use of a batch file. This file moves all needed files to a "Refresh" folder that the testers can access to update the datamodel in their work folder. An example of that file is shown here.

```
@echo off
copy HRS08.bsk V:\2008HRS\Testing08\HRS_CMRefresh\HRS08.bsk
copy HRS08.bpk V:\2008HRS\Testing08\HRS_CMRefresh\HRS08.bpk
copy HRS08.bjk V:\2008HRS\Testing08\HRS_CMRefresh\HRS08.bjk
copy HRS08.bfi V:\2008HRS\Testing08\HRS_CMRefresh\HRS08.bfi
copy HRS08.bdb V:\2008HRS\Testing08\HRS_CMRefresh\HRS08.bdb
copy HRS08.bxi V:\2008HRS\Testing08\HRS_CMRefresh\HRS08.bxi
copy HRS08.bdm V:\2008HRS\Testing08\HRS_CMRefresh\HRS08.bdm
copy HRS08.bmi V:\2008HRS\Testing08\HRS_CMRefresh\HRS08.bmi
echo Copying of files is done!
```

Once the tester is in the HRS_CM system, the program references the master “Refresh” folder and compares the dates of the files with the files in the tester’s own folder. If the two dates do not match, the tester is notified that they may need to “Refresh” (retrieve updated files) for their work folder.



All of this is to ensure that everyone involved remains in sync.

6. Scenario process lifecycle

6.1 Scenario creation and storage

As a first step in building a scenario in the HRS-CM system, the tester sets up an environment for a test by making any needed edits to the preload and checking that they have the latest datamodel. They then open the Blaise Data Entry Program (DEP) and enter keystrokes to correspond with the selected scenario. As a byproduct of the interviewing process, Blaise creates an .ADT file that contains all of the keystrokes required to reproduce that path. The tester can choose to save that ADT file into the Scenario database. To do this a Visual Basic procedure is called that opens that ADT file and loads its contents into a string in memory.

```
ADKFILE = globOutputFolder & "\" & globSampid & "." & globAuditExt
XMLFILE = globOutputFolder & "\" & globSampid & ".XML"
'saving the data stream
Dim wholertext, wholertext2 As String
wholertext = ""
Dim sr As StreamReader
If File.Exists(ADKFILE) Then
    sr = New StreamReader(ADKFILE) 'File IO
    wholertext = sr.ReadToEnd()
    sr.Close()
End If
wholertext2 = ""
If File.Exists(XMLFILE) Then
    sr = New StreamReader(XMLFILE) 'File IO
    wholertext2 = sr.ReadToEnd()
    sr.Close()
End If
```

Once loaded, the procedure connects to our SQL Server database, creates a new record and pushes the string from memory into the database. At this time, several questions are asked of the tester about the scenario being saved and the answers are stored. These questions include requests for a description of the scenario's purpose, a scenario code referencing attributes of the scenario, and a section letter that the scenario is focused on. The procedure also checks for any matching XML preload file in the tester's folder and saves it in the same manner. The code for this procedure is as follows.

```

    Cnn.Open("Driver={SQL
Server};server=SQL_SERVER;database=DevScenario;")
    rs.Open("select * from scenarios order by ScenarioID Desc", Cnn,
    ADODB.CursorTypeEnum.adOpenKeyset,
    ADODB.LockTypeEnum.adLockPessimistic)
    rs.AddNew()
    If wholetext <> "" Then
        rs.Fields("ADKString").Value = wholetext
    End If
    If wholetext2 <> "" Then
        rs.Fields("XMLString").Value = wholetext2
    End If

```

6.2 Scenario retrieval and usage

Now that the preload environment and scenario keystrokes have been saved in the library, several other utilities were needed to allow testers to view descriptions and preload criteria, and to allow recall of a scenario that would satisfy a particular testing requirement. We describe several of these utilities below.

6.3 Scenario Finder

This utility allows the tester to recall a scenario based on specific criteria needed for their test. As noted previously, each scenario has a description, scenario code, and section letter attached to it, all of which allow the tester to quickly find a saved scenario that they wish to re-test.

Each attribute that was saved, such as user name or primary key, is available to the tester from a pull-down menu as a selection criterion. These options are used in the program that recalls the files from the database as shown.

```

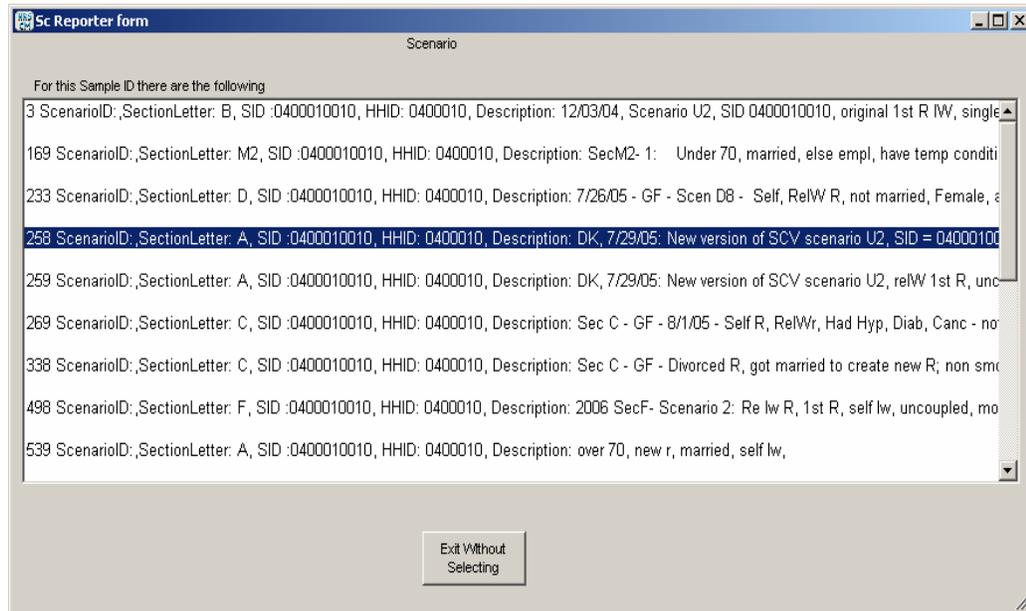
cnn.Open(globCnnScenario)

If globSearchType = SearchType.User Then
    rs.Open("select * from scenarios where CreatorName = " &
    Environment.UserName & " "",
    cnn, ADODB.CursorTypeEnum.adOpenKeyset,
    ADODB.LockTypeEnum.adLockPessimistic)
Elseif globSearchType = SearchType.SID Then
    rs.Open("select * from scenarios where sid = " & globSampid & " "", cnn,
    ADODB.CursorTypeEnum.adOpenKeyset,
    ADODB.LockTypeEnum.adLockPessimistic)
Else
    rs.Open("select * from scenarios", cnn,

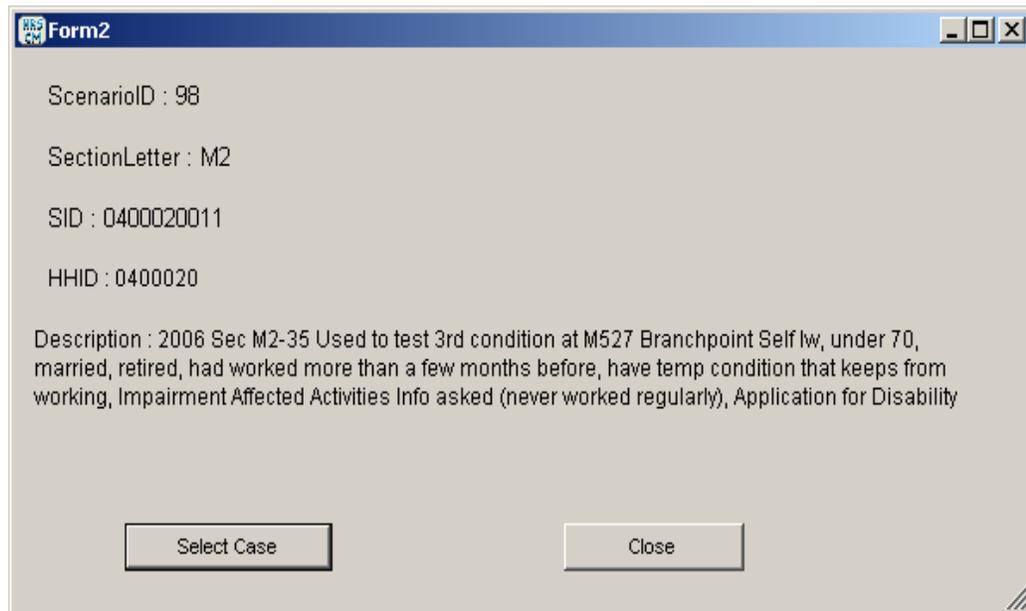
```

```
ADODB.CursorTypeEnum.adOpenKeyset,  
ADODB.LockTypeEnum.adLockPessimistic)  
End If
```

In addition, it is possible to request scenarios based on whether a specified field or fields appear in it. The results are displayed in a form allowing the tester to select a scenario.



By highlighting a record the testers can view the recorded details of the scenario, seen in Form2.



Once a record is selected, the ADT data is pulled into memory, and pushed into a text file, using the appropriate tester's file location. Below is the Visual Basic code that performs that task.

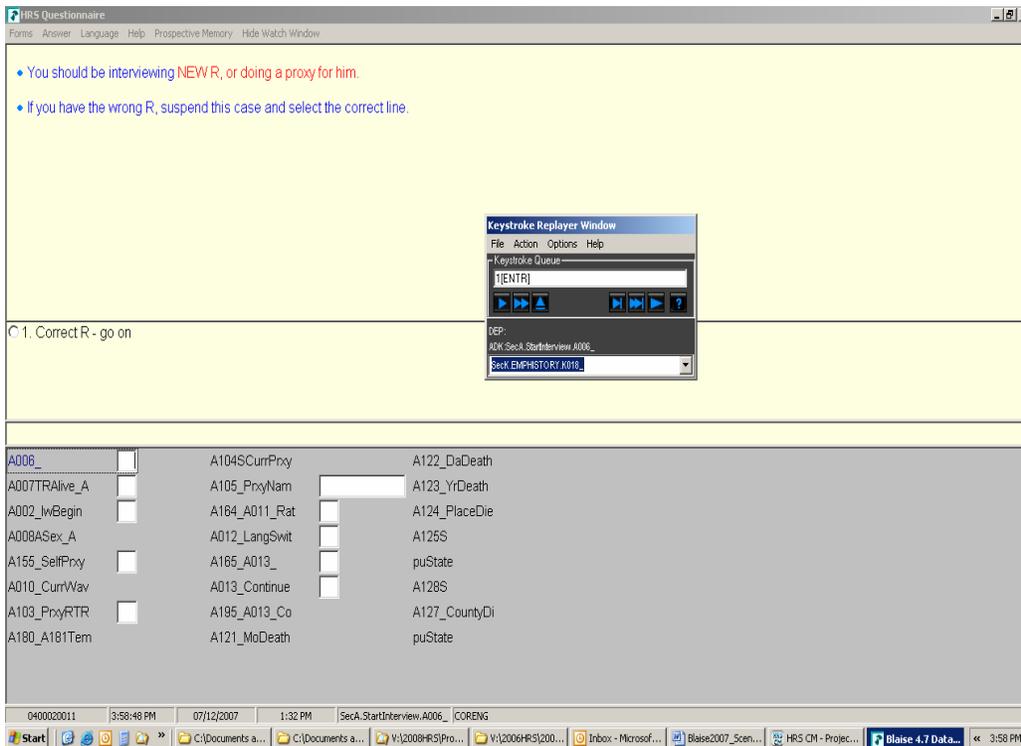
```
ADKFILEout = sAdkLocation & "\" & globSampid & "." & globAuditExt
FileOpen(1, ADKFILEout, OpenMode.Output)
PrintLine(1, rs.Fields("ADKString").Value)
FileClose(1)

If IsDBNull(rs.Fields("XMLString").Value) = False Then
XMLFILEout = sAdkLocation & "\" & globSampid & ".XML"
FileOpen(3, XMLFILEout, OpenMode.Output)
PrintLine(3, rs.Fields("XMLString").Value)
FileClose(3)
End If
```

At this stage, the environment for the case has been recreated and the tester can either restore or replay the case.

6.4 Replayer

The Replayer mentioned earlier in this paper allows a tester to watch the keystrokes be re-applied to the data entry application and interact with the answers to questions, accepting them and moving on, or changing them on the fly.



Above is an image of the keystroke replayer window. The tester is able to replay to any point in the application, stop and alter keystrokes, then continue the interview,

monitoring the effects of their changes. Replaying saved scenarios can quickly ensure that a datamodel change has not affected other areas of the application. This is a major benefit for a questionnaire as large and complex as the HRS.

6.5 Restore

Quite often testers do not need to step through each keystroke individually, but rather need to reach a specific point in the application to re-test a sequence. The “Restore” utility allows the tester to select a location in the application and have all stored keystrokes applied using only the BCP in order to reach that point, without having to spend the time to walk through the entire interview.

7. Outcomes

We have moved from tracking and testing a limited set of scenarios that followed critical paths to being able to test flow and language, replicate data errors from the field, change and retest quickly, and share keystroke files for other testing needs.

Prior to these utilities being developed, we used spreadsheets to document major paths for testing. This technique had some major disadvantages. These became outdated quickly as the instrument was changed. Also, a tester might find a problem and often the programmer would not be able to replicate it, or a tester would not be able to test a path without the ability to change the preload appropriately. Given limited time, we were not able to completely test all modes of our application. As a result of incomplete testing, we often received reports from the field staff that there were problems with flow, screen text, or even “Hard” code errors that would stop the application and cause us to lose valuable field time. In 2004, HRS released nine revisions of the datamodel. In several instances interviewing had to stop while the program was corrected. In 2006, HRS issued only three field revision updates, of which two were planned in advance to update the Spanish language and to change the year reference. The ability for testers to save a critical path and then quickly re-test when changes have been made ensured a much higher degree of accuracy in the data collection application.

These utilities have benefited many parts of the instrument development process. For example, a translator can switch languages at any point in the application to check that the text for a given question is correct. Another benefit of this system is the ability to reproduce problems that are reported from the field. By recovering the .ADK and preload files from the field, we can place them in a tester’s folder and run the same Replayer utilities to replicate the problem. This has proven to be a great time saver for getting quick resolution and turn-around with field issues.

Each section of the application now has a set of twelve or more saved scenarios within the library from each tester. These ensure that every part of the instrument is well tested. We now have the ability to select a variable and scan through the scenarios in the library looking for and reporting on any instance of that variable. This allows testers to find a scenario with the keystrokes that will get them to a specific location in the application. This has proven to be a great timesaver. Also, with this “field finding” ability we can check coverage of the scenarios in the library in relation to our real-time data and know that we are capturing the major paths in our testing.

The benefits go on. The long term effect of this library is that 2006 scenarios will be replayed on the 2008 datamodel after changes and updates have been made. This will ensure that no unintended consequences were introduced as the result of changes or corrections that were applied to some of the more difficult areas of the instrument.

These utilities have made it possible for us to shorten our pre-production cycle, allowing more time for investigators to incorporate the latest innovations. The testing staff has been able to reduce the amount of time they spend on testing and take on other responsibilities.

8. Conclusion

The project staff at HRS aims to produce a reliable and accurate survey instrument that mirrors the complex and continually evolving design of the researchers and to do so in an efficient and timely manner. The story of the development of the tools that HRS uses to achieve this goal is indicative of the complexity of instrument development on a survey of this scale. Over time, what has emerged is a substantial suite of applications that may be instructive for and even flexible enough to be used by other studies. It has certainly proved to be a fertile ground for the development and refinement of new and existing tools at HRS as the example of the Scenario Library described here illustrates. The Scenario Library builds on tools developed previously by combining them to produce a new functionality that has already proved quite valuable. We fully expect this tool to evolve and grow, like the others, according to the results obtained as it is used in practice. Ultimately, through the development of these tools, we aspire to continue to improve the reliability and accuracy of HRS with each successive wave, while becoming ever more flexible and efficient about incorporating new design ideas from researchers even when they are particularly complicated or given on short notice.

CATI MANAGEMENT: Behind the Scenes

David Dybicki and Grant Benson, The University of Michigan¹

1. Abstract

The Blaise CATI management system has a series of built-in features designed to maximize calling efficiencies. However, with declining response rates, it becomes important to be able to further tailor and customize the delivery algorithm to maximize response rates while controlling costs. In this paper we will describe techniques we use programmatically using Blaise manipula scripts in conjunction with Windows task Manager to maximize our sample delivery throughput for several household RDD studies.

2. Introduction

In the face of declining contact and response rates, it is increasingly important to be able to customize and modify delivery algorithms to different populations and study requirements. The Blaise Computer Assisted Telephone Interviewing (CATI) management system is a powerful product created by Statistics Netherlands that provides a basic set of delivery options satisfying most organizational and development needs ‘out of the box.’ However, as has been noted elsewhere (Hart and Bandeh 2003), the vast array of differing organizational needs makes it infeasible for Statistics Netherlands to generate a product that serves the requirements of all organizations. Instead, Statistics Netherlands has opted to create a basic set of features that satisfies most organizational and development needs ‘out of the box’, and allow for a substantial amount of customization through Manipula scripts to meet additional requirements.

This paper describes the implementation of specific Manipula scripts used by The University of Michigan’s Survey Research Center to increase the probability of identifying eligible household sample from a Random Digit Dial (RDD) frame, of obtaining cooperation among resistant households, and of removing sample unlikely to yield completed interviews. In addition, the paper will describe the overnight batch features we use to implement most of these techniques.

¹ The authors wish to acknowledge the significant amount of assistance we have received from Statistics Canada in learning to customize the Blaise CATI management system. Several of the programming techniques described in this paper were developed with Statistics Canada.

3. 'Out of the Box' Settings

The Blaise CATI management system is hard coded to deliver cases that are most likely to yield completed interviews. In particular, the system sorts cases for delivery based on appointment status, such that 'super' appointments are delivered first, followed by 'hard' appointments that have recently reached a busy signal, hard appointments, medium-busy appointments, and so on, down to non-appointment ('default') cases. Within each of the nine priorities, the management system allows additional sorting on virtually any preloaded variable, including time zone, number of dials or calls, and whether a case has had previous contact.

The Blaise CATI management system also includes a fairly flexible tool for rotating cases without contact on the previous attempt through a series of 'time slices'. This is designed to optimize calling by restricting the number of times that an interviewer can make attempts during non-productive times, essentially keeping a case from being delivered during a particular day of the week and time of day once a specified number of attempts have already been made during that time without making contact.

In addition, the Blaise CATI management system contains a number of built-in settings that can be adjusted to allow for routing appointments back to interviewers or groups as well limiting the number of attempts that can be made on cases generally, limiting the frequency of calling cases reaching answering machines, not reaching anyone, or resulting in a busy signal.

3.1 Constraints on the 'Out of the Box' Settings

3.1.1 Missed Appointments

In the event that a hard appointment is missed, either because the respondent was not available, or because an interviewer could not make the call, the CATI management system will reset the case as a medium appointment for the start of the next day. This creates inefficiencies because it crowds out the sample at the start of the day for a time that may not be optimal for calling those cases any way. That is, if the original appointment was for 8 PM the previous evening, there is no reason to assume that calling the case at 9 AM the next day would yield a positive outcome.

3.1.2 Time Slices

By default, time slices work based on the last attempt. If the last attempt resulted in no contact, then the CATI management system will cycle through the user-defined grids until they have all been filled. Once they have been filled, the case is no longer delivered. The problem is that it does not distinguish between cases that have ever had contact on them and cases that have never had contact on them. Particularly if you are concerned about response rates, it is important to continue to make calls on a case once it has been determined that a sample line is eligible or is a working number. Alternatively, it could be helpful to distinguish between types of no contact, for example if one reaches a fax machine or computer noise, which would be potentially indicative of a phone number not being designated to a household phone.

3.1.3 Other Treatment Options

The CATI management system contains special treatment options for several result code types, including days to exclude from the daybatch (for no contact cases and answering machine cases), and number of times that busy cases can be dialed or 'tried'. However, this does not allow the user to distinguish between contact cases such that a resistant case may be treated differently from a general callback, nor distinguish between whether the interviewer left a message on an answering machine or just hung up without leaving a message. These cases potentially have different likelihood of obtaining contact or cooperation, and should therefore ideally be treated differently.

3.1.4 Daybatch Exclusion

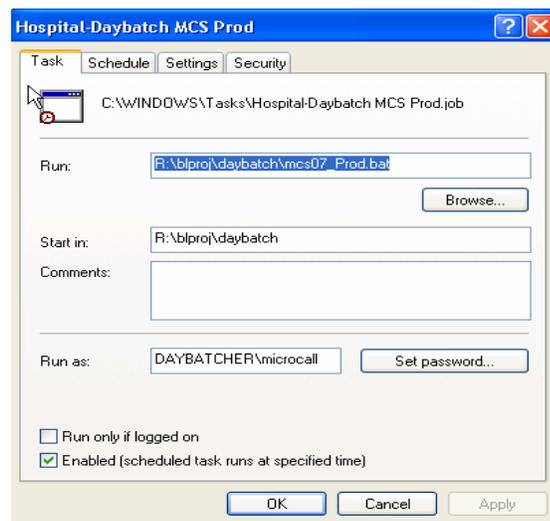
Once total call limits or time slice limits are met, cases are simply excluded from further delivery. This essentially results in the cases being unresolved until the end of a study, unless the delivery parameters are changed, with no indication for why they are excluded.

4. Programming Solutions to Enhance Sample Delivery

In order to make the CATI sample distribution more efficient we updated key variables in our SMS data using Blaise Manipula, and took advantage of many of the pre-existing features, including the various priority levels and appointment types. Most of the programming solutions are implemented as part of an overnight batch process.

4.1 Daybatch Processing

Using Windows task manager we first specify a batch file to run:



The batch file calls a .dat file that contains the following commands to run Blaise hospital, the study specific manipula scripts, the daybatch, and sends e-mails when done:

Code contained in the .dat file:

```
r:\blaise\blaise47\hospital.exe^/Lr:\blproj\daybatch\myproject.log /R
/MR:\blproj\cati\MYPROJECT\PROD\2007_08\SMS.bmi
R:\blproj\cati\MYPROJECT\PROD\2007_08\SMS.bdb

r:\blaise\blaise47\hospital.exe^/Lr:\blproj\daybatch\myproject.log /R
/MR:\blproj\cati\MYPROJECT\PROD\2007_08\MYPROJECT.bmi
R:\blproj\cati\MYPROJECT\PROD\2007_08\MYPROJECT.bdb

r:\blaise\blaise47\hospital.exe^/Lr:\blproj\daybatch\myproject.log /R
/MR:\blaise\cati\sms2006\history.bmi
R:\blproj\cati\MYPROJECT\PROD\2007_08\history.bdb

R:\blaise\blaise47\Manipula.exe^R:\blproj\cati\myproject\prod\2007_08\
ResetHardApts.msu

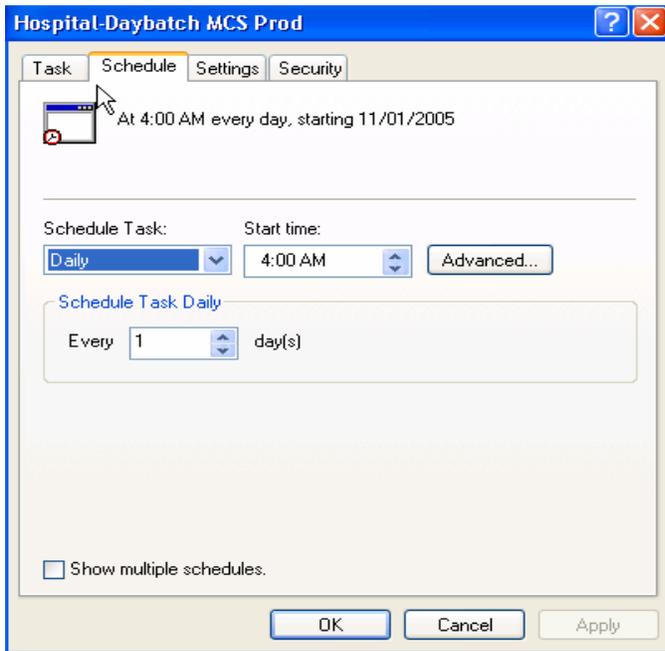
R:\blaise\blaise47\Manipula.exe^R:\blproj\cati\myproject\prod\2007_08\
SetSoftApts.msu

R:\blaise\blaise47\Manipula.exe^R:\blproj\cati\myproject\prod\2007_08\
SetSoftAptsForFax.msu

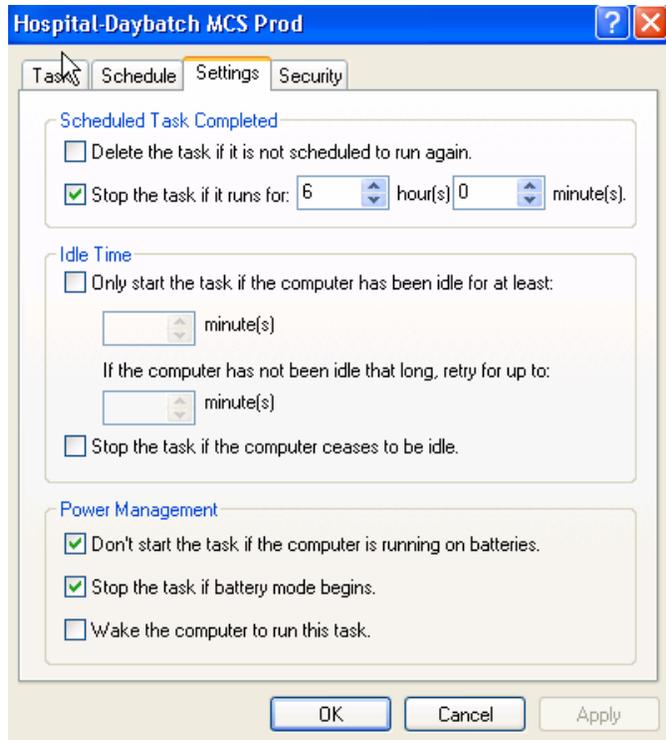
r:\blaise\blaise47\btmana^R:\blproj\cati\myproject\prod\2007_08\sms.bt
s /B /Hr:\blproj\cati\myproject\prod\2007_08\
/Wr:\blproj\cati\myproject\prod\2007_08\

R:\blproj\daybatch\batmail\batmail.exe^R:\blproj\daybatch\batmail\mypr
oject.txt R:\blproj\daybatch\batmail\message.txt
```

Then set time to run, and other settings:



Generally speaking, we try to run the daybatch at the time when it is least likely to interfere with interviewing production. We also limit how long the daybatch can continue to run to ensure that the process has been completed before our interviewing begins.



4.2 Resetting Missed Appointments

The scheduler default for missed hard appointments is inefficient in that when an appointment is missed (for example, the respondent is not home when we call), the case is designated as a medium appointment for the start of the next day when it is unlikely that the interviewer will reach the respondent. To address this issue, we wrote a Manipula script that resets hard appointments to soft appointments for the same general time as the original appointment for the next day. This takes advantage of knowledge that we already have about the respondent in terms of their stated availability, without requiring that interviewers go through and review all the call notes.

The script below checks for the missed hard appointments from the previous day and resets them as a soft appointment for the following day with time ranges (daypart) assigned based on the parameters below. These may be modified based on staff availability or even other historical data that may be available on a case. The key is to take advantage of knowledge that we already have on the contact.

If the missed hard appointment was set for:

- before 1pm
- between 1pm and 5pm
- between 5pm and 9pm
- between 9pm and 10am

It is reset for the next day to:

- a range of 10am - 1pm
- a range of 1pm and 5pm
- a range of 5pm and 9pm
- a range of 9pm and 10am

```

MANIPULATE
  a_counter := 0
  SMSData.OPEN
  SMSData.RESET
  SMSData.READNEXT

  {repeats this operation for every case in the DB}

  REPEAT
    {displays count of missed hard appointments found so far}
    Display('Searching for hard appointments... ' + STR(a_counter))
    {Check for a hard appointment}
    IF ( SMSData.CatiMana.CatiAppoint.AppointType = CertainDate ) AND
      ( SMSData.CatiMana.CatiCall.Regscalls[1].DialResult = 4 ) THEN
      {Check for appointments set in the past}
      IF ((JULIAN(SYSDATE) - JULIAN(CatiMana.CatiAppoint.DateStart)) > 0) THEN
        {Set the appointment type to soft}
        SMSData.catiMana.caticall.Regscalls[1].dialResult := 4
        SMSData.CatiMana.CatiAppoint.AppointType := Period

        {if the missed hard appointment occurs before (and including) 1pm,
        set the soft appointment between 10am and 1pm}

        IF (SMSData.CatiMana.CatiAppoint.TimeStart >= (10,0,0)) AND
          (SMSData.CatiMana.CatiAppoint.TimeStart <= (13,0,0)) THEN
          SMSData.CatiMana.CatiAppoint.TimeStart := (10,0,0)
          SMSData.CatiMana.CatiAppoint.TimeEnd := (13,0,0)
          SMSData.CatiMana.CatiAppoint.DateStart := SYSDATE
          SMSData.CatiMana.CatiAppoint.DateEnd := SYSDATE

          {if the missed hard appointment occurs between 1pm and 5pm set the
          soft appointment between 1pm and 5pm}
          ELSEIF (SMSData.CatiMana.CatiAppoint.TimeStart > (13,0,0)) AND
            (SMSData.CatiMana.CatiAppoint.TimeStart <= (17,0,0)) THEN
            SMSData.CatiMana.CatiAppoint.TimeStart := (13,0,0)
            SMSData.CatiMana.CatiAppoint.TimeEnd := (17,0,0)
            SMSData.CatiMana.CatiAppoint.DateStart := SYSDATE
            SMSData.CatiMana.CatiAppoint.DateEnd := SYSDATE

            {if the missed hard appointment occurs between 9pm and 10am Drop the
            appointment}
            ELSEIF (SMSData.CatiMana.CatiAppoint.TimeStart > (21,0,0) ) AND
              ((SMSData.CatiMana.CatiAppoint.DateStart + 1 = SYSDATE) AND
              (SMSData.CatiMana.CatiAppoint.TimeStart <= (10,0,0))) THEN
              SMSData.CatiMana.CatiAppoint.TimeStart := EMPTY
              SMSData.CatiMana.CatiAppoint.TimeEnd := EMPTY
              SMSData.CatiMana.CatiAppoint.DateStart := EMPTY
              SMSData.CatiMana.CatiAppoint.DateEnd := EMPTY

              {if the missed hard appointment occurs at any other time, the soft
              appoointment is set between 5pm and 9pm}
              ELSE
                SMSData.CatiMana.CatiAppoint.TimeStart := (17,0,0)
                SMSData.CatiMana.CatiAppoint.TimeEnd := (21,0,0)
                SMSData.CatiMana.CatiAppoint.DateStart := SYSDATE
                SMSData.CatiMana.CatiAppoint.DateEnd := SYSDATE
              ENDIF
            a_counter := a_counter + 1

            SMSData.WRITE
          ENDIF
        ENDIF
        SMSData.READNEXT

      UNTIL SMSData.EOF

```

One additional modification may be to route the case back to the interviewer who set the original appointment. In general, if an interviewer has been able to set the original appointment, he or she may have established a positive relationship with the respondent, thereby increasing the likelihood of obtaining an interview. On the other hand, the interviewer may not be available the next day, but the risk can be minimized by giving the deactivation delay a low number of minutes (say, 10 minutes) in the general parameter settings.

The script below looks for same day or future hard appointments, and assigns them to the interviewer who made the appointment:

```

MANIPULATE

SMSData.READNEXT
REPEAT
  {Check for a hard appointment}
  IF (SMSData.CatiMana.CatiAppoint.AppointType = CertainDate) AND
    (SMSData.CatiMana.CatiCall.Regscalls[1].DialResult = 4) AND
    (SMSData.CatiMana.CatiAppoint.DateStart >= SYSDATE) THEN

    { Assign hard appointments to be interviewer who made appointment }
    SMSData.Towhom := SMSData.CatiMana.CatiAppoint.WhoMade
    SMSData.WRITE

  ENDIF
  SMSData.READNEXT
UNTIL SMSData.EOF

```

4.3 Routing Cases Between Multiple Time Slices

The Blaise CATI sample management system allows for setting up multiple time slices. By having a Manipula script determine whether there is ever contact on a case, for example, the time slice field can be updated to move a case from a minimal time slice to a more inclusive time slice.

Often, studies will want to place a limit on the number of consecutive no contact attempts that are made on RDD cases as they are unlikely to yield contact – much less completed interviews – after a certain number of attempts (e.g., Eckman, O’Muircheartaigh, and Haggerty 2004). Thus, it makes sense to force a grid or time slice upon these cases. However, once contact has been established, the project may still find it helpful to route the case through a grid in the event of future no contact attempts, for example if the project wishes to ensure that cases are called multiple times in a day but at specific different times of day, but eliminate the restrictions on how many total attempts may be made.

To accomplish this, two or more time slices would be set up in the CATI Manager. The first time slice would be fairly restrictive, allowing 6 or 12 total no contact tries for example, and the second would be fairly generous, allowing as many as 100 total tries spread out throughout different days of the week and times of day.

Cases would start out in the first time slice. As part of the nightly process, a Manipula script would determine if contact had ever been made on a case, and would then insert the second time slice if that had occurred.

MANIPULATE

SMSData.READNEXT

REPEAT

IF (SMSData.StudyType = RDD) **AND**
(SMSData.Flags.ContactMade = YES) **THEN**

SMSData.Timeslice := 'S-2'

ENDIF

SMSData.WRITE
SMSData.READNEXT

UNTIL SMSData.EOF

4.4 Subsampling and Gridding

Assigning a final status to cases that have gone through grids or time slices can also greatly increase efficiency. Alternatively, projects may also subsample cases after a certain number of calls. This is done to increase efficiency for certain types of sample or to facilitate non-response analysis.

The following script writes an automated call note for a predetermined subset of “Strata 3” subsampling cases whose calls are greater than 9. For the particular project below, Strata 3 cases are RDD cases that could not be reverse matched to any name or address, suggesting that there is a significant likelihood that these are non-household (and therefore non-sample) numbers. By using the Blaise CATI sample management system to automatically code out these cases as a non-interview, we can eliminate potential interviewer or supervisor bias in assigning result codes, and subsequently be able to apply a two-phase sampling strategy.

MANIPULATE

```
SMSData.READNEXT
```

REPEAT

```
{Strata = UV 70 SubSampling = UV 71
  IF (SMSData.UserValue[70] = '3') AND (SMSData.UserValue[71] = '0') AND
    (SMSData.CallRecord.Call_Num + 1 > 9) THEN

  IF (SMSData.CallRecord.ResultCode <> '4201') AND
    (SMSData.CallRecord.ResultCode <> '4202') AND
    (SMSData.CallRecord.ResultCode <> '4203') AND
    (SMSData.CallRecord.ResultType <> 'FI') AND
    (SMSData.CallRecord.ResultType <> 'FR') AND
    (SMSData.CallRecord.ResultType <> 'HO') AND
    (SMSData.CallRecord.ResultType <> 'NI') AND
    (SMSData.CallRecord.ResultType <> 'NS') AND
    (SMSData.CallRecord.ResultType <> 'PR') AND
    (SMSData.CallRecord.ResultType <> 'TR') THEN

    SMSData.ToWhom := 'Home'
    SMSData.Callrecord.Call_Num := SMSData.Callrecord.Call_Num + 1
    SMSData.CallRecord.Call_date := SYSDate
    SMSData.CallRecord.Call_Time := SYSTime
    SMSData.Callrecord.Resultcode := '7002'
    SMSData.Callrecord.ResultType := 'NS'
    SMSData.Callrecord.Call_Note := 'Automated call record by
      system when strata = 3
      Subsampling flag = 0, and
      this is the 10th call'

    SMSData.WRITE

    HistoryData.INITRECORD
    HistoryData.SampleID := SMSData.SampleID
    HistoryData.HistRecord.PhoneNum := SMSData.Phone
    HistoryData.HistRecord.Call_num := SMSData.Callrecord.Call_Num
    HistoryData.HistRecord.Call_Date := SYSDate
    HistoryData.HistRecord.Call_Time := SYSTime
    HistoryData.HistRecord.ResultType := 'NS'
    HistoryData.HistRecord.ResultCode := '7002'
    HistoryData.HistRecord.Call_Note := 'Automated call record by
      system when strata = 3
      Subsampling flag = 0,
      and this is the 10th call'

    HistoryData.HistRecord.WhoPhoned := SMSData.WhoCalled
    HistoryData.HistRecord.ToWhom := 'Home'
```

```

HistoryData.HistRecord.IncentiveAmt := SMSData.CallRecord.IncentiveAmt
HistoryData.HistRecord.ApptStartDate := SMSData.CatiMana.CatiAppoint.DateStart
HistoryData.HistRecord.ApptEndDate := SMSData.CatiMana.CatiAppoint.DateEnd
HistoryData.HistRecord.ApptStartTime := SMSData.CatiMana.CatiAppoint.TimeStart
HistoryData.HistRecord.ApptEndTime := SMSData.CatiMana.CatiAppoint.TimeEnd

HistoryData.WRITE

SetBTHFields
WriteBTH
ENDIF

ENDIF { (SMSData.UserValue[70] = '3') AND
          (SMSData.UserValue[71] = '0') AND
          (SMSData.CallRecord.Call_Num + 1 > 9) }

SMSData.READNEXT

UNTIL SMSData.EOF

```

A similar script can be generated to automatically code out cases that are still in time slice 1 after 12 attempts, for example.

4.5 Other Uses of Appointments

Often projects will need to enable quicker identification of non-sample or non-household telephone sample lines, delay recontact effort on resistant sample, or create controlled follow-up effort for methodological experiments. All of these functions can easily be accommodated in the background using the built-in appointments functions in Blaise.

4.5.1 Faxes

For household telephone surveys, we typically code out telephone lines that are designated for fax machines. One way to validate that a phone number is designated for a fax machine, is to require that the number has never previously had contact on it, and has at least one day time and one evening call reaching a fax machine.

The script below applies to morning cases from the previous day reaching fax machines. The Manipula script checks the contact flag to ensure that there has never been any contact, and if these conditions are met, the Manipula script sets an evening soft appointment, in this case from 6:00 pm to 9:00 pm:

```

MANIPULATE

REPEAT
  IF SMSData.Flags.ContactMade <> Yes THEN
    IF SMSData.CallRecord.ResultCode IN ['1601'..'1603', '2009'..'2010'] THEN
      SMSData.catimana.caticall.Regscalls[1].dialResult := 4
      {Appointment}
      SMSData.CatiMana.CatiAppoint.AppointType := Period
      SMSData.CatiMana.CatiAppoint.TimeStart := (18,0,0)
      SMSData.CatiMana.CatiAppoint.TimeEnd := (21,0,0)
      SMSData.CatiMana.CatiAppoint.DateStart := SYSDATE
      SMSData.CatiMana.CatiAppoint.DateEnd := SYSDATE
      SMSData.WRITE
    ENDIF
  ENDIF
ENDIF
SMSData.READNEXT
UNTIL SMSData.EOF

```

If the evening call results in a fax noise again, a second script runs in the background to code out the phone number as non-sample.

4.5.2 Days to Delay

In order to increase response rates, projects must address how they work with resistant cases. Research indicates that if a case “rests” for a period of time, interviewers are more likely to obtain subsequent cooperation from respondents than otherwise (Triplett, Scheib, and Blair 2001). To ensure that cases did receive the required exclusion from delivery after an initial resistance, we programmed a feature to allow projects to determine how soon a case should be called back. In addition, we assigned delivery of the case such that it would randomly be assigned to different times of day, with a higher probability of returning in the evening.

By allowing managers to set the number of days to exclude resistant cases ‘on the fly’, we were able to accommodate not only changes in study protocols, such as increased incentives, but also approaching the conclusion of the study period. For example, even if it is optimal to wait for two weeks after an initial resistance before attempting recontact, that guideline is useless if the study ends in one week. Additionally, by randomly assigning when the case is redistributed, we avoid overloading one particular shift with these resistant cases.

The following code delays resistant cases by a pre-determined amount of days for a randomly selected appointment time for result codes equal to 4301, 4302, 4303 or 4304 which are our codes for initial resistance:

MANIPULATE

```
ProjectData.GET('SRC.SRO.MyProject)
NumOfDays := ProjectData.ProjDaysToDelay
```

REPEAT

```
IF SMSData.CallRecord.ResultCode IN ['4301'..'4304'] THEN
  SMSData.CatiMana.Caticall.Regscalls[1].dialResult := 4
  {Appointment}
  SMSData.CatiMana.CatiAppoint.AppointType := Period
  SMSData.CatiMana.CatiAppoint.DateStart :=
    SMSData.CallRecord.Call_date + (0,0, NumOfDays)
  SMSData.CatiMana.CatiAppoint.DateEnd :=
    SMSData.CallRecord.Call_date + (0,0, NumOfDays)
  RandomNumber := RANDOM (7)
  IF RandomNumber = 0 THEN
    SMSData.CatiMana.CatiAppoint.TimeStart := (10,0,0)
    SMSData.CatiMana.CatiAppoint.TimeEnd := (13,0,0)
  ELSEIF RandomNumber = 1 THEN
    SMSData.CatiMana.CatiAppoint.TimeStart := (13,0,0)
    SMSData.CatiMana.CatiAppoint.TimeEnd := (16,0,0)
  ELSEIF RandomNumber = 2 THEN
    SMSData.CatiMana.CatiAppoint.TimeStart := (16,0,0)
    SMSData.CatiMana.CatiAppoint.TimeEnd := (18,0,0)
  ELSEIF RandomNumber = 3 THEN
    SMSData.CatiMana.CatiAppoint.TimeStart := (18,0,0)
    SMSData.CatiMana.CatiAppoint.TimeEnd := (20,0,0)
  ELSEIF RandomNumber = 4 THEN
    SMSData.CatiMana.CatiAppoint.TimeStart := (18,0,0)
    SMSData.CatiMana.CatiAppoint.TimeEnd := (20,0,0)
  ELSEIF RandomNumber = 5 THEN
    SMSData.CatiMana.CatiAppoint.TimeStart := (20,0,0)
    SMSData.CatiMana.CatiAppoint.TimeEnd := (21,0,0)
```

```

ELSEIF RandomNumber = 6 THEN
    SMSData.CatiMana.CatiAppoint.TimeStart := (20,0,0)
    SMSData.CatiMana.CatiAppoint.TimeEnd   := (21,0,0)
ENDIF

SMSData.WRITE
ENDIF
SMSData.READNEXT

UNTIL SMSData.EOF

```

Similar code may be written for other procedures, including follow-up on cases where voice messages are left on answering machines, or methodological experiments.

5. Conclusion

Our organization has enjoyed a great deal of success with the Blaise CATI call scheduler thanks to the control afforded by the manipulation of critical data. We have written hundreds of manipula scripts over the years that have enhanced the features of the call scheduler. If we don't get it right at first we continue to tweak the scripts until we get the desired outcome and manipulation of the daybatch. It's through this constant perseverance that efficiency is achieved.

6. References

Stephanie Eckman, Colm O'Muircheartaigh, and Catherine Haggerty, "Effects of Gridout Procedures on Response Rates and Survey Quality." *Journal of Survey Methodology* (2004), pp. 4762-4765.

Leonard Hart and Linda Bandeh, "Blaise CATI at Mathematica." Paper presented at the 2003 International Blaise Users Conference.

Timothy Triplett, Julie Scheib, and Johnny Blair, "How Long Should You Wait Before Attempting to Convert a Telephone Refusal?" *Proceedings of the Annual Meeting of the American Statistical Association*, August 5-9, 2001.

Westat (2004) *Blaise CATI Guide*. Rockville, MD.

Computer-Assisted Dialing: What will it do for you?

Dan J. Bernard, Marketing Systems Group (MSG)

1. Background

Computer-Assisted dialing has been an increasingly integral part of many market research organizations telephone data collection operations over the last 10 years. While being adopted as a means of improving interviewer productivity, an unexpected by-product has been quality improvement. This paper will address the economic and quality issues surrounding the use of computer-assisted dialing – showing the benefits to social research.

With the adoption of computer-assisted dialing by organizations such as NORC and other large social research agencies, it is rapidly becoming accepted in social science research circles.

2. Overview

While the goal of this paper is to review productivity and quality improvements that can be realized by computer-assisted dialing, it will likely be useful to review how a computer-assisted dialer works. Not so much that there are operational misconceptions – it is common that seasoned researchers are quite unfamiliar with how a dialer really works. What it sounds like. How it interfaces with a CATI system. What controls there are to eliminate respondent abuse.

3. Sample Management

Most organizations have determined that it is best to leave the sample management to the CATI system or special front end software developed to do so. Many telemarketing dialers have built-in scripting and number management capabilities, but generally these capabilities are woefully inadequate for the survey researcher's requirements.

Generally, a telephone number or group of telephone numbers is sent to the dialer for dialing. Once dialed, a message is sent back to the CATI or sample management system from the dialer reporting the results of the dialing. In the case of a non-connect such as a non-working number or busy signal, the CATI system will disposition the sample record as such and send a new number to be dialed. All of this is transparent to the interviewer – therefore the dialing of a second number occurs very rapidly – minimizing unproductive time between dialings. When the dialer detects a person answering the phone, the voice is connected to the interviewer and a message is sent to the CATI system to display the introductory paragraph.

The measured accuracy of the automated disposition of the sample is considered a significant feature by one large agency.

4. Methods of Dialing

The different terms used for computer-assisted dialing can be very confusing. Terms frequently employed are: auto, power, predictive, adaptive, super, progressive, preview, etc. There are three fundamental methods of automated or machine-based dialing:

- **Auto** – telephone number is dialed by a dumb modem
- **Power** – dialer can detect and disposition certain dialing results such as non-working numbers
- **Predictive** – dials more than one number per interviewer using sophisticated statistical algorithms and number knowledge base to deliver a live respondent more quickly
- **Preview** – is an option utilized in auto and power dialing which allows the interviewer to review data from the sample record before talking with the respondent.
- **Hybrid** – is an option combining preview and predictive refined by NORC, Pulse Train Ltd. and MSG and named HATIsm. Hybrid-Assisted Telephone Interviewing. It combines the best of both worlds – delivering the high response rates of preview dialing with the efficiency of predictive dialing.

4.1 Auto Dialer

- Dials one telephone number under interviewer control via modem or black box
- Accurate dialing of telephone number
- Dials number much more quickly than manual dialing
- Approximate productivity gains of 3 to 5%
- No abandonment of calls
- No intelligent sensing of dialing result
- No ability to dial “ahead” of the interviewer

4.2 Power Dialer

- Dials one telephone number per interviewer - can be under interviewer control or paced by the programmer
- Builds on all auto-dialer features
- Can automatically detect fax/modem, ring no-answer, non-working, busy signal. Programmatically exchanges messages with CATI system to disposition old number and retrieve new number for dialing
- Conservative productivity gains of 24-50%
- Less tangible benefits of improved working environment
- No abandonment of calls
- No ability to dial “ahead” of the interviewer

4.3.1 Performance Gains - Power

- 1000 23-minute; 94% incidence; Completes per Hour 19% higher
- 950 23-minute; 6.5% incidence; CPH 53% higher
- 2000 10-minute; 95% incidence; CPH 68% higher
- Qualify and transfer to IVR; CPH 100% higher
- Analysis of 1,700,000 dialings: 31 seconds to connect Vs. 56 seconds manually; 43% Reduction
- Large company yields overall 24% increase
- 22 minute; 8% incidence; CPH 96% higher

4.4 Predictive Dialer

- Dials telephone numbers in a ratio greater than 1:1
- Builds on all power dialer features
- Uses sophisticated statistical algorithms to calculate quantity of telephone numbers to dial
- Allows adjustment of call abandonment percentage
- Conservative productivity gains of 25% over power dialing

- Can contribute to respondent abuse via call abandonment if not properly managed. A research sensitive dialer will allow control of abandonment to be held to one in 10,000 dialings.

4.4.1 Performance Gains - Predictive

- Side-by-side comparisons show 25-50% improvement over power dialing with less than 5% abandonment
- With higher abandonment rates- claims run to 300%

4.5 Hybrid Dialing

- Combines predictive dialing with preview dialing
- If a number has never been connected to a potential respondent, it will be dialed predictively.
- If the number has been previously connected, it will be dialed in preview mode.
- This blended solution maintains the high response rates of preview dialing, but introduces the efficiencies of predictive dialing.

4.5.1 Performance Gains – Hybrid Dialing

- Productivity improvements up to 25%

5. What else can computer assisted dialing do for you?

5.1 Can replace need for PBX:

All call centers need to connect to the public switched telephone network. Calls must be silently monitored for quality assurance purposes. Supervisors must have the ability join a call as required. Some operations require the ability to accept inbound calls.

All these features generally associated with a PBX can be performed by many automated dialers, therefore eliminating the need for a PBX and thus saving capital investment on six-figure PBX's.

5.2 The perfect vehicle for call center decentralization:

- Simple integration with VoIP PBX allowing remote interviewers to connect via VoIP back to a central office and then out to the PSTN (public switched telephone network).
- Improved Quality Control
 - Audio recording introduces robust QC toolset.
 - Live monitoring can originate in any location
 - Exhaustive call data is available

5.3 Optional modules provide additional capabilities:

- Remote Audio Monitoring – Clients, project staff or quality control staff can dial in from remote locations to monitor interviewers.
- Digital voice capture of open ends – allows for more exhaustive probing by interviewer. When the respondent realizes interviewers are writing verbatim responses, they tend to self-edit or abbreviate responses. When the computer is recording the response for later coding - a richer, more complete open-end response is obtained.
- Whole interview recording – The ability to audio record an interview can serve many purposes. On large agency sees it as significant tool in its quality control and training efforts.
 - Coaching of the interviewer
 - Dispute resolution
 - Questionnaire development.
 - Integrate with IVR (Interactive Voice Response)
 - Automated inbound/outbound switching, Call blending, Automated Call Distribution to Spanish speaking interviewers, for example.

5.4 Interviewer & Productivity Management

- Enforces standardized call rules
- Eliminates dialing errors
- Faster dialing means greater throughput
- Dialing modes can be assigned on a study by study and/or station basis
- Real-time graphic and tabular reporting of interviewer productivity

- Full silent monitoring capabilities – both local and remote.

5.5 Facilities Management

- Real-time and historical production reporting, by interviewer, study, shift, site, client, and date
- Scheduling module provides information on number of interviewers and supervisors, and those briefed
- Local and remote monitoring capabilities
- Real-time analyses and reporting of trouble on telephone lines

6. How Dialers Interface with CATI

- For example, the MSG system is a 20 slot, industrial strength Intel-PC with special telephony hardware by Dialogic
- E1, T1, ISDN, or CO lines plug into boards inserted into backplane
- Lines from interviewing stations are punched onto demarc block and cross connected to lines going to station boards
- Dialer is connected to CATI server via serial connection or Ethernet using TCP/IP
- CATI system manages sample file

7. What is Heard by the Interviewer?

- Power Mode:
 - Some systems can be set to pass call progress tones to the interviewer or just the respondent voice on connects.
 - The interviewer will usually hear ‘ello’
 - The call will sound like a normal call to the respondent
- Predictive Mode:
 - No call progress tones can be heard
 - The interviewer will usually hear some part of the ‘hello’
 - The call should sound like a normal call to the respondent unless the call is abandoned

8. Research Vs. Telemarketing

- Research has a limited sample frame. The telemarketing supply is comparatively unlimited.
- A primary goal of research is a high response rate
- Researchers cannot afford respondent abuse - on a project OR industry basis
- Predictive dialing works best with more people, researchers often have 5 to 10 people working on a given project

9. What is Research doing Different

- We know more about a given telephone number than anyone in the country
- We pay attention to call history
- We will predict the probability of connection rather than predict when an “agent” will be finished
- We are offering predictive dialing with “near zero” abandonment
- Predicting probability of connection works with just a few interviewers
- We can dial numbers in fractional ratios, e.g. 1:1.7 rather than 1:2 or 1:3 like some telemarketing systems which forces high abandonment rates
- Traditional predictive is an optional setting – as is *research*PredictiveSM
- We have the flexibility to do it many ways: power, probability of connection, traditional predictive
- Use of automated answering machine detection is discouraged. Studies have shown that some 20% of true respondent contacts are misclassified as answering machines.

10. Beyond Productivity – The Improved Environment

Empirically, it is found that new interviewers are successful more quickly when using computer-assisted dialing. Less productive interviewers find it easier to keep up with more productive interviewers. As such, some organizations have concluded that computer-assisted dialing:

- Improves interviewer retention by helping them be successful more quickly
- Makes their job easier, therefore more desirable
- Provides a discipline that isn't innate

- Facilitates improved job satisfaction
- VP large agency:
 - “I much more enjoy running the phone shop.”
 - “The tedious part of the job is gone.”

Other tangential benefits of autodialing:

- Consistent application of dialing technique.
- Automatic and accurate call result disposition.
- Gives the supervisor time to do things other than push for productivity

Overall, these factors help improve project quality.

11. Social Research is Different

- It’s often heard that social research is different – especially when it comes to length of interview – therefore negating the impact of autodialing. Analysis of 1.7 million dialings shows that 70% of interviews are completed on the first connect. This allows social research to benefit from the gains of computer-assisted dialing.
- Social research call rules often call for double and triple the number of dialings done by market research. This actually gives the advantage to social research of being able to make use of computer-assisted dialing. The more dialing of telephone numbers you do, the more productive it can be.
- The adoption of predictive dialing by NORC in their 500 seat call center has, in one fell swoop, legitimized the use of computer-assisted dialers in social research. Two other major social science research organizations in the States are testing and considering computer-assisted dialing.

12. Are Dialers Expensive?

- They are one of the few things in this industry that can demonstrate a return on investment in under a year.
- Are you having trouble finding interviewers?
- Would you like to improve project quality?

Contact Information:

Dan J. Bernard
Vice President
Marketing Systems Group
5000 Central Park Drive, Suite 204
Lincoln, NE 68504 USA
+1-402-489-0000

dbernard@m-s-g.com
www.pro-t-s.com

Using audit trails to monitor respondent behaviour in an Audio-CASI questionnaire

Olivier Bart (INSEE, France)

1. Introduction

From November 2005 to January 2006, Insee and Drees (Department of Health) have conducted the *Événements de Vie et Santé* (EVS) survey under CAPI. After a face-to-face interview dealing with relationships between tragic events of life (death or suicides of close relations, violences, ...) and health, respondents had to answer a brief ACASI¹ questionnaire gathering the most sensitive questions: sexual activity, alcohol and drug use.

We have collected audit trail information for around 9000 complete surveys, which enabled us to consider a statistical study on this data. As it was our first ACASI experience, we chose to focus on the ACASI part to monitor respondent behaviour facing this tool, and try to improve data collection methodology.

This paper will explain how we processed the adt files and present the main results we obtained on the following issues:

- What is the average time spent by the respondent before giving an answer?
- Do respondents replay questions? Do they review or correct their answers?
- Who are the people who give up the ACASI questionnaire before the end?

All aspects will be discussed from the respondent's point of view (age, sex, nationality, academic level) and from the questionnaire's point of view (open/closed-ended questions, questions about dates/facts/feelings ...).

Besides, this paper aims at suggesting concepts for describing respondent behaviour. Perhaps more than the results on our survey, this may give ideas to anyone who would analyse an audit trail file and extract indicators from it. Every definition we suggest here can certainly be improved!

2. Blaise² options for the ACASI

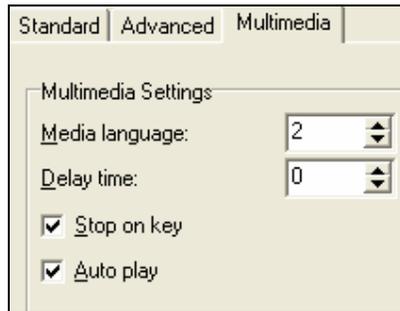
During the first test, the "Stop on key" option was not checked, because survey managers were afraid that people would not listen to the whole question, as question text was always written on screen³. The ACASI questionnaire was far too long and respondents complained a lot for having to wait for the end of the question file. So for the real survey, the "Stop on key" option was chosen, and "Auto play" too. A yellow sticker was stuck on the "F11" key, in order to help people to use the replay function. Each question was followed by the instruction: "If you need to hear the question again, please press the yellow key". As Insee always uses the "Auto enter" option, this concerned the ACASI part too.

¹ Audio Computer Assisted Self-Interview

² The EVS survey used Blaise 4.6

³ In order to increase confidentiality, the ACASI questionnaire of the *Cadre de Vie et Sécurité* survey (2007) makes it possible to hide question texts. In this case, we can imagine that it is harder to answer before the end of the question file...

Figure 1. Blaise multimedia options used for the EVS survey



Besides, to prevent interviewers from being able to read answers given in the ACASI part and thus increase confidentiality for respondents, a locking system was implemented. At the end of the ACASI questionnaire, a locking block was put on the route (and also declared as a parallel block in case of interruption before the end): it asked respondents if they wanted to lock their answers for good and prevent anyone (including regional survey administrators) from reading them. As this question was used as a filter question for the ACASI questionnaire, it became unreachable as soon as the “Yes” answer was chosen.

3. Audit trail files processing

Audit trail files (.adt) are text files recording the sequence of screen and fields visited during a Blaise interview. Every time a field is entered or left, or a specific action is performed, a record containing a time stamp, the current name, value and state of the field is created.

At the moment, Insee does not use adt files for real surveys, but only for tests, on small samples. The EVS survey was an exception, and remains the only one where we have collected so many adt files: 523, one for each interviewer. They contain 9139 forms including at least one variable on the ACASI questionnaire.

As many organisations did, in order to make statistical analysis easier, we had to transform this huge amount of data into a more structured format. We have thus converted the adt files in SAS datasets including, among others, for each line:

- date
- time
- the form's primary key
- the name of the field considered
- its current value

Figure 2. Example of SAS dataset created from an adt file

	Date	Heure	Etat	Identif	Nom	Statut	Valeur	Cause	Action	Comm
3238	16/11/2005	11:41:30	FinChp	21606	VIOLSANT.CONCLUSION.MERCI	NORMAL	1	INPUTLINE FULL		
3239	16/11/2005	11:41:30	EntChp	21606	VIOLSANT.EVSAA.QAA1.AACRSRAG	NORMAL				
3240	16/11/2005	11:41:49	FinChp	21606	VIOLSANT.EVSAA.QAA1.AACRSRAG	NORMAL	23	INPUTLINE FULL		
3241	16/11/2005	11:41:50	EntChp	21606	VIOLSANT.EVSAA.QAA1.AACSNBP	NORMAL				
3242	16/11/2005	11:42:09	FinChp	21606	VIOLSANT.EVSAA.QAA1.AACSNBP	NORMAL	1	NEXT FIELD		
3243	16/11/2005	11:42:09	EntChp	21606	VIOLSANT.EVSAA.QAA1.AACSSXP	NORMAL				
3244	16/11/2005	11:42:16	FinChp	21606	VIOLSANT.EVSAA.QAA1.AACSSXP	NORMAL	1	INPUTLINE FULL		
3245	16/11/2005	11:42:16	EntChp	21606	VIOLSANT.EVSAA.QAA1.AACSPACT	NORMAL				
3246	16/11/2005	11:42:27	FinChp	21606	VIOLSANT.EVSAA.QAA1.AACSPACT	NORMAL	1	INPUTLINE FULL		
3247	16/11/2005	11:42:27	EntChp	21606	VIOLSANT.EVSAA.QAA1.AACSMST	NORMAL				
3248	16/11/2005	11:42:52	FinChp	21606	VIOLSANT.EVSAA.QAA1.AACSMST	NORMAL	2	INPUTLINE FULL		
3249	16/11/2005	11:42:52	EntChp	21606	VIOLSANT.EVSAA.QAA1.AACSPILL	NORMAL				
3250	16/11/2005	11:43:06	FinChp	21606	VIOLSANT.EVSAA.QAA1.AACSPILL	NORMAL	2	INPUTLINE FULL		
3251	16/11/2005	11:43:06	EntChp	21606	VIOLSANT.EVSAA.QAA1.AACSVIG	NORMAL				
3252	16/11/2005	11:43:11	FinChp	21606	VIOLSANT.EVSAA.QAA1.AACSVIG	NORMAL	2	INPUTLINE FULL		
3253	16/11/2005	11:43:11	EntChp	21606	VIOLSANT.EVSAA.QAA1.AACSTES	NORMAL				
3254	16/11/2005	11:43:20	FinChp	21606	VIOLSANT.EVSAA.QAA1.AACSTES	NORMAL	1	INPUTLINE FULL		
3255	16/11/2005	11:43:20	EntChp	21606	VIOLSANT.EVSAA.QAA1.AACSDTES	NORMAL				
3256	16/11/2005	11:44:18	Action	21606	VIOLSANT.EVSAA.QAA1.AACSDTES				REMARK CLICKED	Field/V
3257	16/11/2005	11:44:44	Action	21606	VIOLSANT.EVSAA.QAA1.AACSDTES				REMARK CHANGED	Field/V
3258	16/11/2005	11:44:52	FinChp	21606	VIOLSANT.EVSAA.QAA1.AACSDTES	NORMAL	1980	INPUTLINE FULL		
3259	16/11/2005	11:44:52	EntChp	21606	VIOLSANT.EVSAA.QAA1.AACSDCT	NORMAL				
3260	16/11/2005	11:45:31	FinChp	21606	VIOLSANT.EVSAA.QAA1.AACSDCT	NORMAL	6	INPUTLINE FULL		
3261	16/11/2005	11:45:31	EntChp	21606	VIOLSANT.EVSAA.QAA1.AACSRREST	NORMAL				
3262	16/11/2005	11:46:03	FinChp	21606	VIOLSANT.EVSAA.QAA1.AACSRREST	NORMAL	1	INPUTLINE FULL		
3263	16/11/2005	11:46:03	EntChp	21606	VIOLSANT.EVSAA.QAA2.AAINTRO1	NORMAL				
3264	16/11/2005	11:46:49	FinChp	21606	VIOLSANT.EVSAA.QAA2.AAINTRO1	NORMAL	5	INPUTLINE FULL		
3265	16/11/2005	11:46:50	EntChp	21606	VIOLSANT.EVSAA.QAA2.AAAL12M	NORMAL				
3266	16/11/2005	11:47:03	FinChp	21606	VIOLSANT.EVSAA.QAA2.AAAL12M	NORMAL	2	INPUTLINE FULL		
3267	16/11/2005	11:47:03	EntChp	21606	VIOLSANT.EVSAA.QAA2.AAALSOUV	NORMAL				

After having summarised information regarding each field of each form on the same line (number of times, duration of each time where the field was accessed ...), we obtain one file of 209,364 observations. Our first goal was to study duration on each field, and we chose to focus on the *passage1* variable, which represents duration of the first time spent on the field (not necessarily the longest one, that would have been another possible option). We noticed odd values (up to 2 hours and a half on the same question!), so with the help of the graphical distribution of the *passage1* variable, we chose to exclude durations longer than 360 seconds, which finally gave us 209,333 observations.

4. The respondents behaviour facing the ACASI questionnaire

In each sampled household, one person was drawn to be respondent for the individual face-to-face interview, and then for the ACASI questionnaire at the end. The person drawn was the one who had the nearest following birthday (from the interview date) among the 18-75 years old. Theoretically, this kish method guarantees an equal repartition of men and women and of age groups. Besides, proxy was not authorised: the kish had to answer in person. In order to have results depending on kish characteristics, age, sex, nationality and academic level of the 8893 individuals who had read at least two questions⁴ of the ACASI questionnaire were transferred from the adt file to the SAS database.

⁴ A lot of cases where only one question was read may correspond to a demo made by the interviewer and can be considered as unit non-response.

Then, in order to discuss each issue from the questionnaire's point of view, we have defined the following two groups for each question of the ACASI questionnaire:

- Form (based on technical criteria):
 - **Closed-ended:** the variable has an enumerated type, from the basic Yes/No type up to nine choices.
 - **Open-ended:** in this case, Blaise integer type. This makes a slight difference with the usual definition of "open-ended" which implies an open zone where free text can be typed. For example, we consider the question "During your whole life, how many sex partners did you have?" as open-ended even if the authorised range for answers is in fact limited (1..99).

This distinction makes a real difference for the respondents: with the "Auto enter" option, Blaise goes automatically to the next question when a valid answer is entered, but for open-ended questions, if the respondent enters an answer that does not fill the entire field (just like "3" if the variable type is "1..99"), he needs to press the "Enter" key himself to validate his answer.

- Content (kind of information the questions ask for):
 - **Facts:** question asking for objective realities
Ex: "During your whole life, have you ever drunk alcohol?"
 - **Feelings:** more subjective question, open to interpretation
Ex: "During your whole life, did you ever feel that you should reduce your alcohol consumption?"
 - **Numbers:** objective question, to be answered with a number (except dates)
Ex: "How many glasses of alcohol did you drink last sunday?"
 - **Dates:** objective question, to be answered with a date, an age, or asking if an event happened during a well-defined period
Ex1: "How old were you when you smoke cannabis for the first time?"
Ex2: "Did it happen during the last twelve months?"

4.1. Duration on each field

On average, the ACASI questionnaire followed nearly 68 minutes of face-to-face interview, which was already longer than usual. Most of the time, Insee, in order to ease the interviewers' work, to reduce the response burden and to increase data quality, tries to limit interview time to one hour. In that context, interview time is a permanent concern for a survey manager. In an ACASI questionnaire, it of course depends on the number of questions, but also on the time spent on each question: adt files give us very accurate information on this subject.

With the Blaise options used here, respondents were able to answer before the end of the audio file reading the question: this happens indeed more than 7 times out of 10. So the duration measured on each question with the help of the adt file can be in fact

considered as the sum of three more or less distinct stages: listening and/or reading (and understanding) the question, looking for the answer, typing the answer. The duration of the question file can be used as a reference for measuring the “length” of each question, as every file was recorded by the same voice, of someone speaking very clearly. As we can easily imagine, this contributes a lot to the response time (see model below): a long question takes more time to be understood and answered than a shorter one...

Table 1. Mean response time by question and mean number of questions answered for each part of the ACASI questionnaire

	Mean number of questions answered	Mean response time by question	Field	Mean duration of audio files
ACASI questionnaire	23.3	15.1 s	Respondents to at least 2 questions of the ACASI questionnaire	19.9 s
<i>sex part</i>	10.0	17.5 s	Respondents to at least 1 question of the considered part among those above	23.1 s
<i>alcohol part</i>	10.0	14.1 s		17.9 s
<i>drugs part</i>	3.7	10.8 s		16.5 s

The average response time by question is 15.1 s, median is 12 s and third quartile is 19 s. Drugs part gathered 23 of the 62 questions of the ACASI questionnaire, but most of the questions concerned only those who declared having consumed cannabis or other drugs, which explains the very low mean number of questions answered. Response time increases with age, decreases with academic level, is higher for men, for foreigners⁵ and for open-ended questions.

We have built a logistic model to see how the factors listed below explain the chance to be in the higher quartile of the response time distribution (more than 19 s), and to try to distinguish the different effects. In order to limit the selection bias due to filters based on age and gender of the respondent (giving thus specific populations of respondents to filtered questions), we only kept the 34 unfiltered questions for the model. This excluded in particular the drug part of the analysis because it concerned only the 18-64 year olds. In spite of this precaution, model results have to be interpreted cautiously because there may remain bias due to the internal filters of the questionnaire: if a filter question is strongly explanatory for the response time, this could upset the model. Introducing these filter questions in the model could be a solution, but the questionnaire we are working on is very complicated, so in order to keep the model readable, we will admit this simplification here.

This model underlines two very strong effects: the duration of the sound file and the form of the question have a decisive influence on the response time. For the length of the question, it was obvious. For the open-ended questions, we can suggest two explanations: first, with equal sound file durations, a closed-ended question is answered more quickly because the respondent answers as soon as he hears the right proposition for him (let’s recall here that all answers were read in the sound file for categorical variables); secondly, an open-ended question requires more thinking and more typing (and even sometimes manual validation as we said before). All other effects are significant too (p-values < 0,0001).

⁵ The ACASI questionnaire was only in French for the EVS survey

Table 2. Characteristics affecting the chance to get a long response time for a question

Characteristics		%	Difference with the reference situation (point)	Odd Ratio
Respondents	18-24 years old	20.0	-1.0	0.8
	25-34 years old	21.5	-0.9	0.9
	45-54 years old	27.7	+1.2	1.2
	55-64 years old	29.9	+2.2	1.4
	65-75 years old	32.5	+3.0	1.5
	No qualification	32.6	+1.2	1.2
	CEP (Primary education certificate)	33.2	+0.9	1.1
	Brevet (Year 10 / Ninth grade)	26.8	-0.8	0.9
	Bac. tec. (Technological baccalaureate)	23.5	-1.4	0.8
	Bac. gen. (General baccalaureate)	21.6	-2.3	0.6
	Bac + 2 (2 years after baccalaureate)	21.8	-2.2	0.6
	Bac > 2 (more than 2 years after baccalaureate)	20.3	-2.8	0.5
	Foreigner	31.8	+2.0	1.3
Naturalised French	30.1	+1.4	1.2	
Male	27.8	+1.2	1.2	
Questions	Sound file \geq 20 s	46.7	+41.8	13.5
	Form: open-ended	42.6	+51.0	19.5
	Content: dates	25.7	-4.1	0.4
	Content: numbers	43.7	-3.1	0.5
	Content: feelings	2.4	-2.9	0.5
	Alcohol part	16.8	-2.6	0.6
Whole population		26.4	-	-
Reference situation		6.5	0	1

- Independent variables: age, higher diploma, duration of the sound file, question form, nationality, question content, part of the questionnaire, gender.
- Reference situation: a closed-ended question, asking for facts, belonging to sexuality part of the questionnaire, read by a sound file lasting less than 19 s, answered by a woman, aged between 35 and 44, native French, having a vocational training certificate or a vocational diploma.
- Field: unfiltered questions, belonging to ACASI questionnaires where at least 2 questions were read, and where every independent variable of the model is known, which makes 158 752 questions.
- Interpretation: 20.0% of the questions read by the 18-24 year olds have a response time in the upper quartile (more than 19 seconds); belonging to this age group makes the value decrease by 1.0 point (or multiplies by 0.8 the chance of having a response time exceeding 19 seconds) compared to the reference situation, all other characteristics being equal (statistically significant at 5%).

4.2. Replay, review and correct

The respondent's ability to use the Blaise tool was an important concern for EVS survey managers. They had no idea of how people who had never used a computer during their whole life would react to this tool. First tests were quite encouraging, and interviewers were even able to convince people who began by refusing the ACASI questionnaire. They used to stay nearby while the respondent was answering the ACASI questionnaire, which enabled them to help if necessary. Adt files give us

here the opportunity to see how respondents have used the Blaise tool. We will define the three following concepts to describe the respondent behaviour.

- Someone **replays** a question file when:
 - we have a “Mediastart” action in the adt file,
 - this action occurs at least 2 seconds after the previous one when several are recorded in a row.

Actually, we sometimes have dozens of mediastart actions recorded in the same second. This may correspond to continuous pressures on the F11 key. We could have kept only one mediastart action for each time the respondent is on the variable, but we think that repeated mediastart actions are significant, so we chose the above definition.

- Someone **reviews** an answer when:
 - the field has already a value when it is entered,
 - at least 1 second is spent on the field,
 - the field has always the same value when it is left.

This may happen because of the “Auto enter” option: as soon as a valid value is entered on a field, Blaise goes automatically to the next field on the route. If you want to verify your previous answer, you need either to look at the formpane (but who knows except the professional interviewers?) if it is still visible, or to jump back to the previous question: this is the phenomenon we are measuring here. The 1 second criterion was added to eliminate the variables you will find “on your road” leading to the question you really want to reread.

- Someone **corrects** an answer when:
 - the field has already a value when it is entered,
 - the field has a different value when it is left.

This may correspond either to a mistype, or to a regret on a given answer.

First important result: in nearly 60% of the forms, neither of the three phenomena can be found. This means that in those 60% of forms, the ACASI questionnaire is filled without any replay or jump back. On the opposite, we have 3% of forms cumulating the three. At the form level, replaying at least one question is observed in 24.5% of the cases, reviewing in 16.9% and correcting in 15.4% of the cases. Reviewing at least one question and correcting at least one question are strongly linked phenomena: both require the same ability to use arrow keys from the respondent.

In order to analyze these phenomena at the field level, we will define the following indicators for each variable:

$$\text{Replaying intensity} = \frac{\text{Number of replays of the sound file associated with the variable}}{\text{Number of ACASI forms where the variable is read at least once}} \times 100$$

$$\text{Reviewing intensity} = \frac{\text{Number of variable reviews}}{\text{Number of ACASI forms where the variable is read at least once}} \times 100$$

$$\text{Correcting intensity} = \frac{\text{Number of variable corrections}}{\text{Number of ACASI forms where the variable is read at least once}} \times 100$$

The mean replaying intensity is 2.1, reviewing intensity is 3.6 and correcting intensity is 1.0. They vary a lot from one question to another, revealing interesting effects due to the questionnaire design. We will not discuss these results further here because we would need to include the whole questionnaire to show relevant examples. We can just notice that correction intensity is very interesting to study for very sensitive questions such as drug consumption questions: do respondents correct their first answers? If so, what kind of corrections are they doing?⁶

Besides, we have built three logistic models, with the same principles and same comments as the former one, in order to explain the chances of having at least one replay, review or correction on a question. They show that higher age, lower academic level and foreign nationality tend to increase the three phenomena. Replays, reviews and corrections happen really more often on open-ended questions: does this mean that open-ended questions cause more difficulties to respondents? We tend to believe it...

4.3. The item non-answer issue

As all fields were programmed with the NOREFUSAL, NODONTKNOW and NOEMPTY Blaise attributes, item non-answer corresponds here with interrupted questionnaires. Obtaining the highest response rate as possible is always a major concern for survey managers, so we tried to know where in the questionnaire some respondents gave up and what characteristics may explain interruptions.

The interruption rate among questionnaires where at least 2 questions have been read (which excludes cases of “hidden” unit non-answer) is 2.2%. More than 3 interruptions out of 4 concern questionnaires where less than 4 questions were read: they happen at the very beginning of the questionnaire. This is quite comforting from the questionnaire design point of view, because it means there is no specific rejection linked with a question or a theme: the people who gave up would probably have done so anyway.

In order to determine the effects leading to a higher chance of interruption, we have built a logistic model including the respondent’s characteristics and the duration of the preceding face-to-face interview.

⁶ For results on these subjects, see Bart (2007), « Analyse du comportement des répondants lors de la collecte du questionnaire auto-administré de l’enquête Evénements de Vie et Santé », *Mémoire FCDA, ENSAI*.

Table 3. Characteristics affecting the chance to have the ACASI questionnaire given up before the end

Characteristics	%	Difference with the reference situation (point)	Odd Ratio
55-64 years old	2.9	+ 0.9	2.2
65-75 years old	6.6	+ 2.7	4.8
No qualification	5.6	+ 1.2	2.6
CEP (Primary education certificate)	5.0	+ 0.7	2.0
Foreigner	6.5	+ 1.4	3.0
Whole population	2.2	-	-
Reference situation	0.7	0	1

- Independent variables: age, higher diploma, nationality, sex, duration of face-to-face interview.
- Reference situation: a woman, aged between 35 and 44, native French, having a vocational training certificate or a vocational diploma, who had a face-to-face interview duration in the 2nd quartile.
- Field: respondents to at least 2 questions of the ACASI questionnaire, with every independent variable of the model known, which makes 8893 individuals.
- Interpretation: among respondents aged between 55 and 64, 2.9% gave up the ACASI questionnaire before the end; belonging to this age group makes the value increase by 0.9 point (or multiplies by 2.2 the chance of having the ACASI questionnaire given up) compared to the reference situation, all other characteristics being equal (statistically significant at 5%).

We have three significant effects: ages over 55, low academic level and foreign nationality increase the non-response risk. This is especially true for the 65-75 year olds: they had a shorter questionnaire than others because the drug part was only for the 18-64, but belonging to this age group multiplies the risk by nearly 5 anyway. However, a 6.6% rate of interruption is not that bad, but the over 65 remains clearly a population to be considered with an extreme care for the ACASI questionnaires.

We did not have here the languages fluently understood by the respondent, but we may imagine that some of the foreign people had difficulties to understand French. That is indeed what the interviewers reported. Gender of the respondent and duration of the preceding face-to-face interview have no significant effect: interruptions do not seem to be due to weariness caused by an especially long face-to-face interview.

5. Conclusion

Adt information is able to tell a lot about respondent behaviour. The fact that all the phenomena we have described and measured here remain limited is in my opinion good news for data quality: it would have been quite worrying to see a lot of corrections or interruptions on a variable. The results we obtained here may help questionnaire designers in their constant quest to reduce data collection time, for example by privileging closed-ended questions. Two specific populations appear in every analysis: over-55 years old and foreign people. Technical solutions exist for helping both of them. For example, the *Cadre de Vie et Sécurité* survey includes now an ACASI questionnaire too, and several languages have been implemented under the Blaise tool (English, German, Arabic, Turkish, ...). For the over 55, nothing particular has been done yet, but we can imagine a specific design (larger font size?) that would perhaps help achieving a better response rate. ACASI technology is now recognized as very efficient for collecting sensitive data, so we will certainly

experiment with it again in the next years, and try to improve our programming methods on this specific aspect.

6. Acknowledgements

Many thanks to Jean-Yves Bart for his help in English translation.

7. References

Ali J. (2003), « [Data quality monitoring using the Blaise audit trail](#) », *SSC Annual Meeting, Proceedings of the Survey Methods Section*.

Bart O. (2007), « Analyse du comportement des répondants lors de la collecte du questionnaire auto-administré de l'enquête Événements de Vie et Santé », *Mémoire FCDA, ENSAI*.

Bumpstead R. (2001), « [A Practical Application of Audit Trails](#) », *Proceedings of the 7th International Blaise Users Conference*, pp. 302-307.

Burrell T. (2003), « [First steps along the Audit Trail](#) », *Proceedings of the 8th International Blaise Users Conference*, pp. 183-190.

Gatward R. (2001), « [Audio-CASI with Challenging Respondent](#) », *Proceedings of the 7th International Blaise Users Conference*, pp. 220-231.

Hansen S.E., Marvin T. (2001), « [Reporting on Item Times and Keystrokes from Blaise Audit Trails](#) », *Proceedings of the 7th International Blaise Users Conference*, pp. 320-335.

Ho P., Smith L., Chan W., Kang S., Tan L. (2005), « [SASifying CAPI Audit Trails for Analysis](#) », *2005 Federal Committee on Statistical Methodology Conference Papers*

Lapierre B., Meyer S. (2006), « [Using the Audit Trail Data To Evaluate the Quality of Collection of the Canadian National Longitudinal Survey of Children and Youth](#) », *2006 Joint Statistical Meeting Abstract Book*, pp. 234.

Liu Y., Galvan E., Cheung G. (2004), « [Blaise AT Report System](#) », *Proceedings of the 9th International Blaise Users Conference*, pp. 240-250.

Penne M. A., Snodgrass J., Barker P. (2003), « [Analyzing Audit Trails in the National Survey on Drug Use and Health \(NSDUH\)](#) », *Proceedings of the 8th International Blaise Users Conference*, pp. 155-174.

Experiences Using an Event History Calendar in the Panel Study of Income Dynamics

April Beaulé, Eva Leissou and Youhong Liu, The University of Michigan

1. Introduction

The Panel Study of Income Dynamics (PSID) is a nationally representative longitudinal study of approximately 8000 U.S. families. The Board of Directors of the PSID continually search for ways to innovate the data collection instrument. For the 36th wave of the PSID, a major new component was added. A childhood health calendar was programmed to collect information on the most common childhood medical conditions including asthma, diabetes and allergies. Since this childhood health component is retrospective focusing on the time from the respondent's birth to age seventeen, the design team anticipated that using a calendar module would help trigger the respondent's memory. The PSID had prior experience using an electronic calendar for employment data outside of the main Blaise interview module. This separate employment calendar presented many obstacles, the most significant of which was the processing of separate file structures: Blaise and Access databases. In order to alleviate some of these processing issues, the decision was made to incorporate a Visual Basic (VB) calendar that would be called from the Blaise Data Entry Program (DEP). The data from the VB calendar application was written directly back to the Blaise bdb using a Dynamic Link Library thus avoiding the creation of multiple datasets.

In addition to data structure and extraction issues, the calendar mode versus the standard question list presented us with a number of challenges during interviewer training. This paper describes the challenges that we faced when fielding this calendar including data structure, extraction, and interviewer training as well as the strategies used to mitigate these issues.

2. Background

The impetus for the use of the Event History Calendar method of interviewing surfaced for the PSID when budgetary constraints forced the project to move from an annual interviewing cycle to an every other year interviewing cycle. The primary concern was whether respondents would be able to provide the same level of reporting accuracy for the two calendar years prior to the interviewing year. Based on the cognitive recall work of Robert Belli, the PSID conducted an experiment in 1998 on the use of an Event History Calendar to collect data in the core domains used in the PSID including housing moves and employment spells. Half of the cases were randomly assigned to each condition: (1) Standard question-list format and (2) Event History Calendar format. The results of the experiment showed that the EHC (Event Calendar History) method of interviewing provided more accurate autobiographical retrospective reporting in comparison with the standard question list method. The experiment also showed that there was no significant difference in the amount of time it took to conduct each of the interviewing treatments.¹

¹ From PSID website see overview of Calendar Methods Study
<http://psidonline.isr.umich.edu/Data/documentation/ehc/PSIDcalendarMethodsStudy.html>

Based on the results of the experiment, the leadership of the PSID decided that the core questions regarding employment spells, time off and housing moves would move to an electronic calendar instead of traditional question list. The PSID application was overhauled in 2003 with the two major changes being a move from Surveycraft to Blaise and creating an EHC for the job sections. Since Blaise did not offer a method for programming a calendar type grid, an EHC for the employment domains was programmed in Visual Basic with a Microsoft Access database as the backend. At the time the 2003 application was programmed, there were significant limitations in using Dynamic Link Libraries the most critical of which was stability. The University of Michigan programmers experienced several problems trying to implement alien routers and procedures in large applications such as the PSID.² As a result of these limitations, the PSID application was split into three separate Blaise applications and two Visual Basic calendars. An in-house “Multiple Application Interface (MAI)” was developed to handle moving data from one application to the other.

3. Prior Experience with EHC Data Collection

The data collection using the EHC seemed to go smoothly with the exception of minor technical difficulties for some cases using the utility that passed pre-load to each instrument. The interviewer feedback indicated that the EHC seemed to aid respondents with date recall as the calendar allowed data entry to be flexible. The grid format allowed the movement from one domain and back again depending on how the respondent was able to recall date information. However, the processing and release of data in the first wave using the EHC was difficult and time consuming. Because the PSID is a panel survey, the need for consistency over time is crucial. The employment data was collected in a different format from all previous waves but the final output variables needed to line up with prior waves.

Below is an example of the differences between data collection techniques and output variables. The screen shots show a comparison of how work weeks were asked between the question list and the EHC:

² See 2003 IBUC paper by Hagerman and Kannan, The University of Michigan

Figure 1: Number of weeks and hours worked in Question-List Format

BC42a

EMPLOYER: BLOCKBUSTER (June 2006 to Feb 2007)

How many weeks out of the year did you actually work on this job in 2006, not including any time off that you told me about earlier?

- If R says: "Every week except what I took off", ENTER '97'

Head PY Weeks Worked

BC43

EMPLOYER: BLOCKBUSTER (June 2006 to Feb 2007)

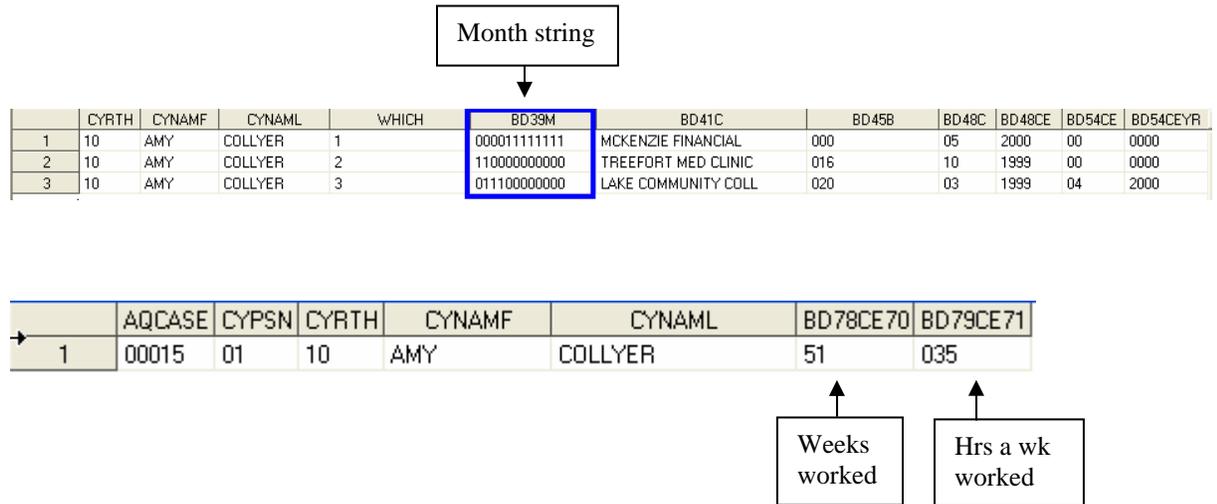
On average, how many hours a week did you work on this job in 2006?

Head PY Weeks Worked

2006 Hours Per Week

In the question-list structure the interviewer marks which months the respondent has worked. This translates into a month string field (Figure 2), one of our key employment variables used during editing. We also ask "So how many weeks out of the year did you actually work at this job 2006?" and "On average, how many hours a week did you work on this job in 2006?" (Figure 1)

Figure 2: Month String, Weeks Worked and Hours Worked- SAS dataset-List Format



In the EHC structure for 2003, we no longer asked the questions in the same way. We asked when each job started and stopped and the interviewer marked each job on the calendar.

Figure 3: Number of weeks worked in EHC Format

 Employment Q-list

BC4. I'd like to know about all of the work for money that you have done for the past two years, from January 1, 2005 to the present. Please include self-employment and any other work that you have done for pay. Start with any job that you had during this time.

BC6. When did you start and when did you stop working for this employer? Please give me all of the start and stop dates if you have worked for (this employer/yourself) more than once.

BC5. What was the name of this employer? [IF NECESSARY: This information will help us to process employment information you gave us. The name itself will never be released as part of data from the study]

[IWER: If no employer name is given, ask for job title or anything that can help identify the job.]

[IWER: Before exiting timelines, probe for any other work for pay, no matter how small.]

R 05 SPR 05 SUM 05 FAL 05 FAL 05 WIN 06 SPR 06 SPR 06 SUM 06 FAL 06 FAL 06 WIN 07 SPR
 R APR MAY JUN JUL AUG SEP OCT NOV DEC JAN FEB MAR APR MAY JUN JUL AUG SEP OCT NOV DEC JAN FEB MA

Landmark Events

Residence

Employment Summary

Not Working

Landmark Events Residence Employment Not Working TimeAway

Employment Data Entry Window

R 05 SPR 05 SUM 05 FAL 05 FAL 05 WIN 06 SPR 06 SPR 06 SUM 06 FAL 06 FAL 06 WIN 07 SPR
 R APR MAY JUN JUL AUG SEP OCT NOV DEC JAN FEB MAR APR MAY JUN JUL AUG SEP OCT NOV DEC JAN FEB MA

Not Working Summary

Smith Brothers

Parmalat

Craftworks

Job# 4

Starting Time				Ending Time					
Year	Season	Month	3rd of Month	Year	Season	Month	3rd of Month		
Before-DK	SUM	AUG	First	2005	WIN	DEC	Last	Current	

Employer: Craftworks

Note:

Figure 4: EHC Table Structure in Access

SampleID	JobN	cDate	SYear	SSeason	i_sMonth	SMonth	Sby3rd	EYear	ESeason	i_eMonth	EMonth	Eby3rd	Employer	Current
0004003	1	07/01/2006	2005	SPR	4	APR	First	2006	SUM	7	JUL	First	Smith Brothers	0
0004003	2	04/21/2007	2006	SUM	7	JUL	Last	2007	SPR	4	APR	Last	Parmalat	1
0004003	3	12/21/2005	B-DK	SUM	8	AUG	First	2005	WIN	12	DEC	Last	Craftworks	0

In the MS Access database (Figure 4) we captured start and stop dates for each employment spell. We had to construct month strings. For respondents with jobs that were seasonal, it was not clear from the start and stop dates alone which months they were actually working.

We also had to construct number of weeks worked and this was tricky for several reasons. The calendar is divided into thirds of months and it was often difficult to tell if a job ended in the second third of June what that meant. Perhaps it meant June 15th or perhaps the 3rd week of June. When jobs overlapped, calculating total weeks and hours worked was even more problematic. The result of the new data collection technique required several weeks of hand corrections by editors in order to construct variables that were comparable to those asked in the question-list format in prior waves.

The incorporation of the EHC also proved a challenge in other areas. Visual Basic is not a robust survey software and thus we were unable to enforce the type of constraints that are available in software such as Blaise. Consistency checks were written for the calendar but for some cases we still ended up with missing data. The in-house application that passed preload from one application to the other also failed at times, which required us to do callbacks on a small number of cases.

One of the major implications of incorporating the EHC was that we were forced to break up the PSID application into five different applications and five different datasets. From the interviewing perspective, this was a significant drawback because once each application was complete; the interviewer could not back up to that section of the interview to make a correction.

For processing and documentation, separate datasets in different formats required us to have different utilities to import data to SAS. The audit trail on the calendar was not as sophisticated as the audit trail in Blaise. While we could generate detailed timing reports from Blaise data, we were not able to replicate these for the calendar data. Automated tools for documentation such as the Michigan Questionnaire Documentation System could not be used on the EHC files. In addition, the EHC files did not contain the same meta-data information that we routinely extract from the Blaise files.

4. Training Interviewers on the use of an EHC

Interviewer trainings start with an introduction to the General Interviewing Techniques (GIT) that teach interviewers core interviewing skills including standard interviewing protocols for all interview components. The teaching and use of standard interviewing protocols is meant to promote consistency across data collectors and is easy to use when the questionnaire format is a question list. The interviewer reads the questions verbatim and records answers, either close or open ended. The EHC sequence of the interview administration has some deviations from this standard because not every question or probe is scripted, and recording answers is done using both mouse and keyboard entry. In addition, the interviewer has to navigate through the various tabs where the scripted questions are or where they record answers.

The EHC administration has presented us with two challenges which we had to consider when developing the training protocols and material: 1) use of non-scripted questions or probes, and 2) recording answers using a mouse and keyboard. For the first challenge we had to teach interviewers that the kind of questions or probes that are allowed based on the fields they have to fill out. For example, in the "Residence Domain", we ask for residential moves done in the last two years, but do not include every question asking for start and end time of each residence move reported. The interviewer uses their own simple phrases or questions to get the year or month of the move. In addition, if the respondent does not know the exact move date, the interviewer probes for the season

when the move occurred. For data entry in Blaise, interviewers are trained to use the keyboard only, but within EHC they are required to switch from keyboard to mouse. Training using multiple data entry modes has been a time consuming task for those interviewers who are less adaptable.

The EHC training has two modes of presentation: home study and in-person training. The goal of presenting the material in different modes is to reach every type of learner, those who are visual and those who need more hands on practice. Interviewers are given a home study packet which includes a study manual, and a DVD. The study manual covers in detail the goals of the EHC questionnaire and an overview of the various tabs, using screenshots to demonstrate the various tasks interviewers will be doing. This manual is a training tool and user's manual. The DVD also provides an introduction to the basic concepts covered in EHC as well as a demo on data entry.

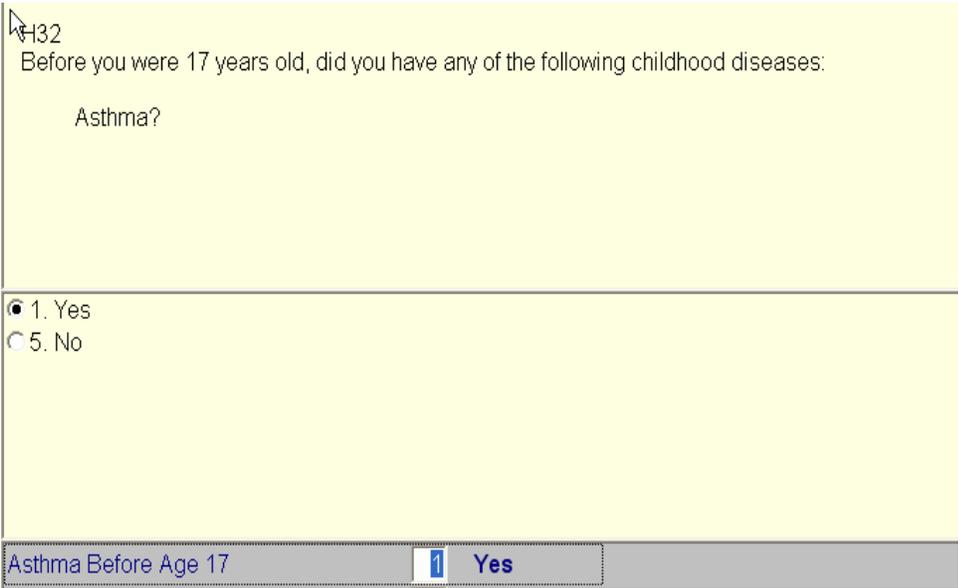
At the in-person training the same material, basic questionnaire concepts and data entry, are covered in more detail. This is followed by practice sessions using a stand-alone module. The stand-alone application allows interviewers to practice the EHC without having to go through the full questionnaire. This module is used for extra practice during the training sessions, the extra help sessions during the in-person trainings, and during the production phase when remedial training may be necessary for some staff.

The interviewer certification protocol includes scoring performance in EHC administration. The certifiers make observations and score specifically the ability of interviewers to do accurate data entries, maneuver with efficiency through the various tabs, and the use of appropriate probes to clarify responses in order to capture a complete and accurate answer.

5. Experimenting with EHC using Dynamic Link Libraries

In spite of the technical difficulties incorporating the calendar, PSID senior staff recognized the value of using an Event History Calendar for certain types of question series. For the 2007 data collection wave, a new retrospective health history sequence was planned. The questions focused on childhood health conditions. This retrospective health history data collection was a prime candidate for a calendar format since the question series focused on a period of the respondent's life from birth to age seventeen. In order to overcome some of the past problems with EHC collection and processing, a DLL call was used that passed control to the subroutine (Visual Basic calendar) if the respondent confirmed they had any of the childhood health conditions listed. If any of the conditions were endorsed in the screening section (Figure 5), then control would pass from Blaise to the EHC where follow up questions were asked. If no conditions were endorsed by the respondents Blaise would continue on to the next field on route.

Figure 5: Screening Question in Blaise for the EHC Childhood Health Calendar



432
Before you were 17 years old, did you have any of the following childhood diseases:

Asthma?

1. Yes
 5. No

Asthma Before Age 17 1 Yes

The data collected in the calendar was then sent back to the Blaise bdb via the DLL. By incorporating the calendar in this fashion, we were able to overcome several issues. Because communication between Blaise and the calendar is dynamic, interviewers were able to back up to make corrections, then move forward and have those changes be reflected in the calendar. For the data processing team, instead of having to process different datasets, all calendar data was written back to bdb. The use of the DLL was significantly more stable than passing preload using our in-house application interface software. We had no reports from the field of the DLL link to the calendar failing.

Given our past experience with employment calendar data, the structure of the health history calendar data was designed for ease of processing. Instead of having start and stop dates like the employment section, we programmed a grid that included a cell for each age from birth to age seventeen. The interviewer began by asking a short series of "landmark" questions about other childhood events. The respondent's answers could then be used to anchor the follow-up health questions (Figure 6).

Figure 6: Landmarks EHC Childhood Health Calendar

PSIDEHC

Person: CHRIS, Respondent-Male Head, Age: 38, DOB: 5/27/1970

If you moved during your childhood, please tell me how old you were at the time of each move, starting with the first move. Any other moves?

Age	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17+
Parental Separations	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17+
Residential Moves	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17+
School Changes	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17+
Asthma	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17+
Allergic Condition	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17+

Buttons: Save, Complete, Exit

Next the interviewer asked follow up questions about the conditions endorsed in the previous screening section (Figure 5). When the interviewer clicked the start and stop cells and then 'Yes', the ages included in that range would be marked '1' meaning 'Yes' (Figure 7).

Figure 7: Simple Data Entry Childhood Health Calendar Conditions

PSIDEHC

Person: CHRIS, Respondent-Male Head, Age: 38, DOB: 5/27/1970

At what age were you first diagnosed with an Allergic condition? Until what age did you have it?

PROBE: "If you don't know the exact age, please give us your best guess."
 PROBE for age using landmarks, starting with the earliest landmark
 For age at which condition ended: if R says "Still have it" or "Older than 16", CLICK "17+" and enter age

Age	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17+
Parental Separations	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17+
Residential Moves	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17+
School Changes	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17+
Asthma	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17+
Allergic Condition	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17+

Buttons: Save, Complete, Exit

We also allowed for more complex coding if the respondent could not remember exact start or stop ages. For example, if a respondent said that he or she had an allergic condition that stopped at age six but could not remember when it started the interviewer could right click on age six and choose 'DK Start' (Figure 8). It terms of data, we record '8' meaning 'Don't Know' for all the ages from birth to age five and then '1' meaning 'Yes' for age six. This method of using one cell to represent each year of childhood made the processing of these variables clean and consistent. We can easily generate start and stop ages for various conditions or we can leave the construction of those variables to the user.

Figure 8: Complex Data Entry Childhood Health Calendar Conditions

PSIDEHC

Person: CHRIS, Respondent-Male Head, Age: 38, DOB: 5/27/1970

At what age were you first diagnosed with an **Allergic condition**? Until what age did you have it?

PROBE: "If you don't know the exact age, please give us your best guess."
 PROBE for age using landmarks, starting with the earliest landmark
 For age at which condition ended: if R says "Still have it" or "Older than 16", CLICK "17+" and enter age

Age	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17+
Parental Separations	0	1	2	3	4	5												
Residential Moves	0	1	2	3	4	5								13	14	15	16	
School Changes	0	1	2	3	4	5								13	14	15	16	
Asthma	0	1	2	1	1	1												17+
Allergic Condition	8	8	8	8	8	8												

Cancel DK Start
 Yes DK End
 DK Refused
 DK All Denial

6. Programming Logistics

The first step in programming the interface was to create the data structure in Blaise to work with the grid in the Visual Basic form. Blaise code was then written to call the Childhood Health Calendar (Figure 9).

Figure 9: Design the Data Structure in Blaise

```

AsthmaEHC : ARRAY[0..17] OF 1..7    {Corresponding to a row in the VB form}
AsthmaEHCAgeGR17 : 17..120        {Store age if grid 17+ is checked}
IF HadAsthma = YES THEN
    ConditionCount := ConditionCount + 1
ELSE
    {This code is necessary here, in case the condition is changed from yes to no.
    The DLL will not clean up the data.}
    FOR I:= 0 to 17 DO
        EHC.OtherProbEHC[I] :=EMPTY
    ENDDO
    EHC.OtherProbEHCAgeGR17 := EMPTY
ENDIF
IF ConditionCount > 0 THEN {At least one problem is checked "yes"}
    EHC.PSIDEHC (xDOB , 'Jone Smith')
ENDIF
    
```

The Visual Basic form is programmed to work with the Blaise data and bdb structure.

Figure 10: Design the Visual Basic Form

Age	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17+
Parental Separations	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
Residential Moves	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
School Changes	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
Asthma	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17+
Respiratory Disorder	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17+

The childhood health conditions are loaded dynamically based on the yes/no screening questions in the prior Blaise DEP sequence. For example, in Figure 10, the respondent endorsed two conditions: asthma and a respiratory disorder. Screening questions for all the other conditions were not endorsed and therefore they do not appear on the grid.

Once the interviewer has completed the grid and has clicked the "Save" button, the data are saved back to the bdb (Figure 11).

Figure 11: Save Data in the Visual Basic Form back to the Blaise bdb

```

If Controls(lblstring)(i).BackColor = Controls(lblstring + "txt").ForeColor Then
  If Controls(lblstring)(i).Caption = "1" Or _
    Controls(lblstring)(i).Caption = "7" Then
    dbp.Field(FieldBasename + "[" + CStr(i) + "]").Text = Controls(lblstring)(i).Caption
    xmlTS.WriteLine "<Field name="" + _
      FieldBasename + "[" + CStr(i) + "]" _
      + "" value="" + Controls(lblstring)(i).Caption + "" />"
  ElseIf Controls(lblstring)(i).Caption = "8" Then
    dbp.Field(FieldBasename + "[" + CStr(i) + "]").Status = blfsDontKnow
    xmlTS.WriteLine "<Field name="" + _
      FieldBasename + "[" + CStr(i) + "]" _
      + "" value=""DK"" />"
  ElseIf Controls(lblstring)(i).Caption = "9" Then
    dbp.Field(FieldBasename + "[" + CStr(i) + "]").Status = blfsRefusal
    xmlTS.WriteLine "<Field name="" + _
      FieldBasename + "[" + CStr(i) + "]" _
      + "" value=""RF"" />"
  Else
    dbp.Field(FieldBasename + "[" + CStr(i) + "]").Text = ""
  End If
End If

```

In addition to saving the data to the bdb, this code (Figure 11) also saves the data to an xml file for backup. The xml file is essential because the Blaise audit trail function does not capture any data collected in an external application such as the VB form.

Figure 12: Load data from bdb to Visual Basic Form

```

If dbp.Field(FieldBasename + "[" + CStr(i) + "]").Status = blfsResponse Then
    Me.Controls(lblstring)(i).BackColor = Controls(lblstring + "txt").ForeColor
    Me.Controls(lblstring)(i).Caption = dbp.Field(FieldBasename + "[" + CStr(i) + "]").Text '1 or 7
ElseIf dbp.Field(FieldBasename + "[" + CStr(i) + "]").Status = blfsDontKnow Then
    Me.Controls(lblstring)(i).BackColor = Controls(lblstring + "txt").ForeColor
    Me.Controls(lblstring)(i).Caption = "8"
ElseIf dbp.Field(FieldBasename + "[" + CStr(i) + "]").Status = blfsRefusal Then
    Me.Controls(lblstring)(i).BackColor = Controls(lblstring + "txt").ForeColor
    Me.Controls(lblstring)(i).Caption = "9"
End If

```

When the calendar is invoked for the first time, all cells in the VB form are empty. However, if the interviewer backs up and makes a correction and revisits the field that launches the calendar, the VB form will reopen for additional edits. The code above (Figure 12) loads the bdb values to the VB cells so that any data collected previously will not be lost.

Figure 13: Consistency Check when Exiting Calendar

```

CHECK  HR4aCheck < 18 AND
        HR4bCheck < 18 AND {HR4bCheck = 18 means all items for Asthma array = EMPTY}
        HR4cCheck < 18 AND
        HR4dCheck < 18 AND
        HR4eCheck < 18 AND
        HR4fCheck < 18 AND
        HR4gCheck < 18 AND
        HR5aCheck < 18 AND
        HR5bCheck < 18 AND
        HR5cCheck < 18 AND
        HR5dCheck < 18 AND
        HR5fCheck < 18 AND
        HR5gCheck < 18 AND
        HR5hCheck < 18
INVOLVING (HR12_INTRO)
"Calendar not Complete!"

```

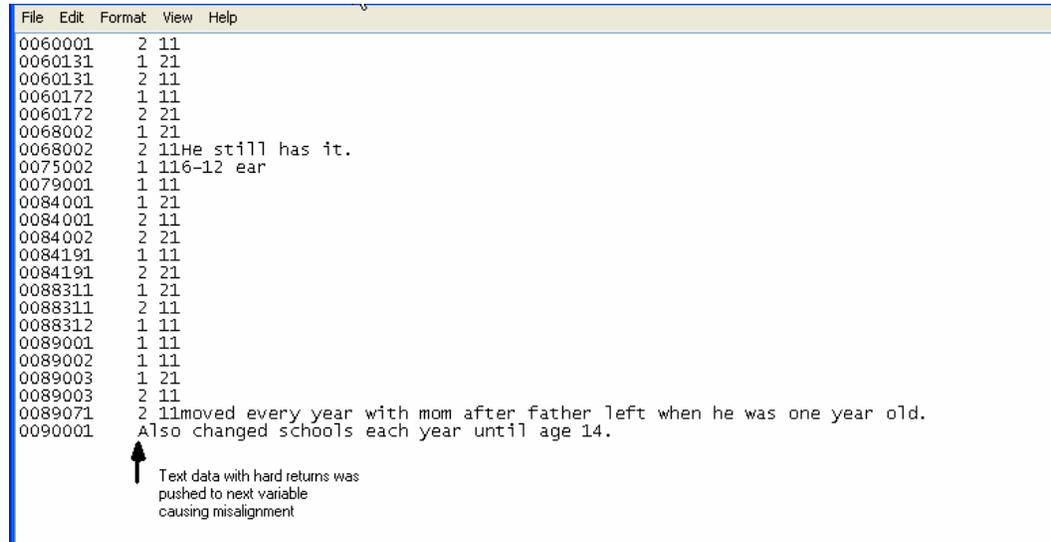
Once the interviewer has indicated that the calendar is complete, a final consistency check (Figure 13) is performed to ensure that all the conditions endorsed have at least one cell filled out indicating when the respondent had that condition. If the interviewer fails to record a value for any condition they are prompted by a hard check indicating the calendar is not complete.

7. Calendar Data Processing

Prior to production, the health history calendar was pretested and we did process and review all pretest data without incident. However, in early data dumps during production, we began seeing some problems in the calendar variables. We researched the cases involved and found that the bdb data looked correct but that our SAS data was not

lining up correctly. We narrowed the problem down to our Blaise to SAS extraction routine. Normally the first step in our routine from Blaise to SAS is to generate an ASCII delimited file with each case representing one row (Figure 14).

Figure 14: Blaise to ASCII delimited file



```
File Edit Format View Help
0060001 2 11
0060131 1 21
0060131 2 11
0060172 1 11
0060172 2 21
0068002 1 21
0068002 2 11He still has it.
0075002 1 116-12 ear
0079001 1 11
0084001 1 21
0084001 2 11
0084002 2 21
0084191 1 11
0084191 2 21
0088311 1 21
0088311 2 11
0088312 1 11
0089001 1 11
0089002 1 11
0089003 1 21
0089003 2 11
0089071 2 11moved every year with mom after father left when he was one year old.
0090001 2 11Also changed schools each year until age 14.
```

↑ Text data with hard returns was pushed to next variable causing misalignment

We incorporated a note field in the EHC health history calendar for the interviewer to provide us any clarification on the calendar (Figure 7- top right corner). Unlike data entry in Blaise, the note field in the Visual Basic calendar allowed interviewers to enter notes that contained hard returns. In Blaise if a hard return is entered in a string field the interviewer is pushed to the next question on the route. Our extraction routine didn't anticipate hard returns in string fields and interpreted the presence of hard returns as an indicator to move the remaining text data to the next variable. For those cases that had hard returns, each instance would push the remaining text data to the next variable and thus all the variables for that point forward were misaligned. Since we hadn't had any cases with hard returns in the note field during pretest, we did not find this problem at that stage.

Our solution was to alter the DLL to remove all the hard returns before writing them back to the Blaise bdb. The new DLL was then sent out to all the interviewer laptops to correct all future cases and was also run on the master file to resolve all of our older cases.

8. Conclusion and Summary

The PSID use of a DLL routine to call an external program such as the EHC was much more successful than our earlier experiences with EHC data collection using a separate application. There are many advantages to having all the data in one format and in one database. The DLL performance was extremely stable in moving data to the external application and back to the Blaise bdb. Having one dataset also means that we can take advantage of other applications that can read Blaise files such as the Michigan Questionnaire Documentation System (MQDS).

Because of our experience with EHC for employment questions, we were able to construct the EHC for the health history calendar with a more concise data structure. It required the use of more variables but reduced ambiguity for start and stop dates. Because of the resulting data structure, users can now easily construct their own variables without PSID staff assistance.

We did find however, that because the EHC is written in another programming language, we need to be careful about how we write back that information to the Blaise bdb. We need to consider the type of restrictions and requirements that are associated with Blaise fields we are writing to and apply the same restrictions and requirements to the fields in the corresponding external application.

9. References

From PSID website see overview of Calendar Methods Study. Retrieved May 31st 2007 from World Wide Web:

<http://psidonline.isr.umich.edu/Data/documentation/ehc/PSIDcalendarMethodsStudy.html>

Hagerman, J. and Kannan, H. (2003): The "Multiple Application Interface" with Blaise and Visual Basic, Proceedings of the 8th International Blaise Users Conference, Copenhagen, Denmark, May 2003.

Retrospective Data Collection

Elizabeth Hacker, Kate Cox, Carli Lessof and Colin Miceli
National Centre for Social Research (NatCen)

1. Objectives of the paper

This paper will outline the selection, design, and use of a Computer Assisted Personal Interviewing (CAPI) event history calendar (EHC) to collect retrospective life course data. In contrast to traditional linear questioning models (Q-lists), the event history approach encourages the sequential and parallel retrieval of information from autobiographical memory, and has been found to produce better-quality retrospective reports (Belli et al 2001)¹. This method was employed in 2007 to collect detailed information about key events that had occurred in the lives of approximately 9,000 participants from the English Longitudinal Study of Ageing (ELSA). We aimed to enhance our understanding of how early life events influence the circumstances of older people by asking participants to recall events that had happened prior to joining the ELSA panel study in 2002. Key dimensions covered were relationships and fertility; housing and mobility; jobs and earnings; and health.

The following aspects of the event history calendar will be described in the paper:

- Advantages of an event history approach over traditional questioning methods in terms of improving recall and increasing user-friendliness
- Outline of the time units and domains covered in the calendar
- Issues relating to the design of the calendar - e.g. flexibility of the interview, changing responses during the interview, displaying events on the calendar
- Implications for interviewer training

2. Background to ELSA and the life history interview

The English Longitudinal Study of Ageing (ELSA) is a study of people aged 50 and over and their younger partners. ELSA explores the dynamic relationships between health and functioning, social networks and participation, and economic position as people plan for, move into and progress beyond retirement. ELSA has been developed through a collaboration between University College London, the Institute of Fiscal Studies, and NatCen, with specialist advice provided by academics at the Universities of Cambridge, Nottingham, East Anglia and elsewhere. Funding for the first four waves of the study was provided by the US National Institute on Aging, and a consortium of British Government Departments.

The ELSA sample was drawn from households that had previously responded to five years of the Health Survey for England (HSE) between 1998 and 2003. The ELSA panel

¹ R.F. Belli, W.L. Shay, F.P. Stafford (2001). Event History Calendars and Question List Surveys: A Direct Comparison of Interviewing Methods. *Public Opinion Quarterly* 65:45-74. American Association for Public Opinion Research.

are interviewed every two years; the first wave of fieldwork was conducted in 2002-2003, with wave two in 2004-5 and fieldwork has recently be completed for Wave 3. A total of 12,009 interviews were conducted at Wave 1. Wave 2 included a visit by a qualified nurse which captured a wide variety of health measures, including biometric measures such as lung function, blood samples, and blood pressure and physical performance tests to assess the respondents' strength and balance.

Wave 3 included a separate follow up interview to collect life history information. Before this, the majority of information collected for ELSA respondents has been about the circumstances of their lives from the time they were first interviewed for HSE until the present day. At HSE, all the ELSA respondents were over 46 years old and some of them were already in their nineties. As a result, we know little about what happened earlier on in their lives. Many aspects of early life have been shown to have a significant impact on people's health, economic circumstances and quality of life in later years. The life history interview enabled the collection of more detailed information about important events that have occurred in ELSA respondents' lives and gives us more of an idea about what their childhood was like. The life history interview collected data in a number of different areas including relationships and fertility; housing and mobility; jobs and earnings; and health. This data will be used to enhance our understanding of how early life and events throughout life have influenced the circumstances of older people.

2.1 Rationale for using the EHC for ELSA

The literature has shown that collecting accurate information about all these different types of events over a lengthy period of time is a challenge². People do not always accurately recall events from the past and life/event histories are often incomplete (i.e. gaps, unsystematic recall of parallel events) and inconsistent across different life domains (i.e. information given on one aspect of their life is inconsistent with another). The Event History Calendar (EHC) method has been developed to help improve the way we ask people about events in their past. This method utilises our understanding of memory processes in order to help people remember past events more accurately (Belli et al, 2001).

3. Background to the EHC approach

3.1 Key elements of the EHC approach

The aim of the EHC is to serve as a device to help respondents recall and anchor changes in major life events. The EHC approach does this in two key ways: first, by providing the respondent with personalised cues and stimuli which may aid recall; secondly, by increasing the flexibility of the interview to allow respondents to recall events in their own way. The key elements of the EHC are outlined below:

A. Providing cues and stimuli to aid recall:

These 'cues' take the form of two types of events or 'landmarks' that the interviewer can use to stimulate the respondent's memory.

² Sudman, S., Bradman, N.M. and Schwarz, N. (1996) *Thinking about answers. The application of cognitive processes to survey methodology* Jossey-Bass.

(i) Personal events

Personal events are ‘internal landmarks’ that are particular to a given individual –such as when they moved house, changed jobs or had children. By plotting life events on the calendar, the EHC approach enables respondents to cross-reference certain life-events with others (e.g. "when I had my first child I was living in house B").

(ii) External events

The calendar shows important ‘external landmarks’ – that is major world or national events such as JFK’s assassination, which may help respondents recall the timing of personal life events which the interview aims to collect.

B. Increasing the flexibility of the interview:

There are four main ways in which a calendar interview is more flexible than a traditional interview (or questioning model):

(i) Entering/inputting events whenever raised

Information can be entered onto the calendar whenever a respondent’s memory is prompted and events are recalled. These events may not be recalled in a sequential order and may relate to different domains or topics. The calendar therefore needs to be flexible enough for events to be recorded in the relevant domain and in the order recalled, rather than being prescribed by the interviewing instrument.

(ii) Moving forwards and backwards in time

The EHC allows for the forward recall of events (i.e. in a chronological order) and/or for respondents to report events starting from the present day and working backwards.

(iii) Order of topics

Respondents are able to choose to answer the topics that they remember easily first. This may help them later in the interview when trying to remember events that are more difficult to recall as the more salient events can be used as recall aids to help anchor the memory.

(iv) Verifying and revising data collected during the interview

As more events are recalled, inconsistencies or errors may become apparent. For example, by checking across domains it may be obvious that a date entered for a job change was incorrect as it was not in the same year as a change in residence, even though the new job was in a new geographical area. The EHC approach allows the interviewer to check responses against those already given and revise the information entered if necessary.

3.2 Layout of the ELSA EHC

The EHC method is in the form of a calendar, which shows time across the top and multiple rows down its side which make it possible to record different kinds of events in respondents' lives (e.g. where they lived, family events). Each column represents a calendar year, going back to the respondent's birth, and each row represents a key domain in a respondent's life, such as births and deaths of children, and changes in accommodation and employment. The calendar is filled in by the interviewer, who conducts a semi-structured interview in which the respondent is asked to review the calendar and report the dates when key events occurred. As respondents answer questions about key life events, these events are recorded on the EHC. Using the EHC technique has been shown to improve the accuracy of the information people can remember³.

Picture 1 shows the ELSA EHC. The top half of the screen resembles a standard Blaise questionnaire with a box where the question wording appears, response categories and response field. The EHC takes up the lower half of the screen. During the interview, questions are asked of the respondent and their responses are entered by the interviewer in a standard way. However, when the interviewer enters the date of an event in the respondent's life, a coloured bar appears on the calendar in the appropriate years (columns), and domains (rows). The personal events box at the bottom right hand side of the calendar displays the information relating to the coloured bars. The box on the left shows important external events (i.e. national and world events) for each year. The information displayed in both these boxes depends on the column that the interviewer's cursor is focused on. The interviewer (and respondent) can only see the events in the respondent's life and external events that happened in one year at a time. Interviewers are able to navigate around the calendar using keyboard arrow keys so that information relating to different years of a respondent's life can be seen.

The questionnaire was written in Blaise, so the Blaise datamodel provides the questions, rules and checks. The "dll" takes the place of the normal data entry program and provides the calendar display and the other normal features of the data entry program, for example, the display of the question text, retrieval of the next question on route and the display of check and signal messages.

4. Development of the calendar

4.1 Stages of development

The ELSA EHC was developed over 31 months (August 2004 to February 2007) incorporating a number of pre-tests and pilots. The aims of this lengthy developmental stage were two fold:

- (a) to ensure that the interview incorporated the elements of the EHC approach; and
- (b) to ensure the EHC was as easy to use as possible (see section 5).

³ Belli, R.F., Lee, E.H., Stafford, F.P. and Chou, C. (2004) 'Calendar and Question-List Survey Methods: Association between Interviewer Behaviours and Data Quality' *Journal of Official Statistics*, Vol 20, No. 2, pp. 185-218.

At each stage of the development of the ELSA life history interview we held discussions with interviewers after the stage was complete to assess the effectiveness of the interview. At some stages the interviews were also evaluated using respondent and interviewer de-briefing questionnaires and by audio-recording the interviews themselves.

The information obtained from these techniques was used to improve the EHC at each stage.

There were three development stages:

(i) PAPI & CAPI pre-test

The EHC approach was initially tested using a paper calendar and CAPI interview. 4 interviewers conducted interviews with 18 respondents. The interviewer firstly asked respondents for the dates of key events from their life and entered these onto the paper calendar. The interviewer then entered these dates into a typical Blaise program whilst intermittently asking respondents follow up questions on the CAPI about these events. The interview at this stage was time intensive and involved duplication of effort with data being entered on the paper EHC and then copied into the CAPI program. However, the pilot did show that the paper calendar was an effective tool in helping respondents remember key events and so this element of the interview was retained.

(ii) CAPI life history calendar I.

It was decided that the whole EHC interview would be conducted in CAPI, in order to address the problems identified at stage (i). However, as we wanted to retain all the key elements of the calendar approach (outlined in section 3.1), this version of the interview was very flexible: it did not involve Blaise type modular interviewing but allowed interviewers to navigate around the EHC using a mouse. In the second pre-test, which involved three interviewers and three respondents, a number of problems with the flexibility of the CAPI interview became apparent. Due to the flexibility of the program the questions about each event were separate which meant that many questions were repeated and that the questionnaire did not flow well (see Section 5.2.1). Interviewers also had problems in using the program due to its flexibility.

(iii) CAPI life history calendar II.

The final development phase included a pilot, where 28 respondents were interviewed by five interviewers, and a dress rehearsal, involving 11 interviewers and 58 respondents.

In response to the problems experienced in the pre-test, the interview was made more structured and incorporated more elements of a standard Blaise interview. The program combined a typical Blaise script with the EHC so that the questions would be generated by the script and input in the calendar. It was at this stage that the program moved to the format where only half the screen was devoted to the calendar. This was a step away from the previous approach where the EHC was central: both for entering information (questions would be launched by clicking in the appropriate year of the calendar) and for displaying the information gathered (so that personalised cues could be generated).

The end result was a less flexible EHC, now combined with a Blaise questionnaire.

5. Flexibility Vs usability of the interview

The next section goes on to explain in a little more detail, the decisions made surrounding the ELSA EHC, some of the difficulties faced, and the rationale for those decisions. It will take each of the key elements outlined in section 3.1 and discuss the extent that each aspect of the EHC approach could be incorporated into the ELSA life history interview. In instances where a more standardised approach restricted the flexibility of the EHC method, an attempt will be made to evaluate whether this reduced the effectiveness of the calendar as a tool to aid recall.

A key development aim of the ELSA life history interview was to ensure the correct balance was achieved between the flexibility of the interview and the usability of the instrument. As outlined in sections 3.2.1 and 3.1, some of the existing literature shows that the EHC approach depends on flexibility to allow respondents to recall events in the way that is most effective for them. For example, respondents should be able to choose the order of topics, and recall events in chronological or reverse chronological order. However, the life history calendar must also be a user-friendly tool. Another consideration was that more than two hundred interviewers would be trained to use the EHC to collect over 8000 respondent life histories.

The development phase showed that there were some aspects of the EHC method which were difficult to apply to a large-scale project. To a degree, it was necessary to restrict flexibility to ensure a user-friendly and workable program was created.

5.1 Providing cues and stimuli to aid recall

5.1.1. Personal events

The personal events box (shown in picture 1) provided a highly visible and easy to use stimulus for both interviewer and respondent. This was an integral part of the interview as it gave the opportunity for interviewers to provide personalised cues when needed, at any point in the interview. The personal events box displays key information recorded during the interview, such as residence, job title and births of children, for each appropriate year. In the CAPI life history interview the events occurring in a particular year were displayed (once the respondent had relayed this information) when the cursor was placed in the appropriate column of the appropriate year.

Feedback from interviewers and respondents (both during the development stage, and since the life history interview has been in the field) have confirmed that this was a useful tool, and is used frequently throughout most interviews.

5.1.2 External events

Putting the EHC onto CAPI allowed for more extensive use of external events because there were less space constraints. External events included important national and world events that were provided to aid respondents' recall and help them anchor personal events. A Microsoft Access database was used to store these events, and the events were displayed on the screen. During the pilot respondents had the opportunity to choose the type and number of external events that could be displayed (e.g. sporting events and/or key political events and/or key events in popular culture). However, the number of

events was in fact reduced before the dress rehearsal to produce one streamlined list that would not overwhelm respondents.

Selecting a list of external events to use in the program which is relevant to respondents and likely to stimulate a response for many individuals is challenging. Additionally, while feedback does show that the inclusion of external events in the CAPI program did lighten the mood of the interview and help maintain respondent interest, on the whole, interviewers felt that these external events did not assist respondent recall.

5.2 Increasing the flexibility of the interview

5.2.1 Entering events whenever raised

A benefit of a paper calendar is the ease of entering events at the time they are raised. For example, if a respondent gives information about a past job, they may then recall where they were living at or around that time. The interviewer can then easily enter this information there and then even though it relates to a different domain and perhaps a different time period.

The first CAPI version of the EHC retained this flexibility. Interviewers could use a mouse and click in the appropriate domain and year where an event had been recalled. This would launch a question box with a series of questions corresponding to the event (e.g. what the event was, start and end dates of the event).

There were several problems with this method. Firstly, interviewers who are trained in traditional linear questioning models had difficulty navigating around a very unstructured program. There was a strong possibility that events could be missed out if they were not recalled in any topic or time order, especially as this made it difficult to identify and check gaps and overlaps. It was also difficult to make the questions flow well, and often there was unnecessary repetition of questions because each event was treated as a separate entity rather than a series of events occurring sequentially. For example, for each event it was necessary to ask both the start and end date. This made the interview more lengthy than necessary and interviewers and respondents found this irritating.

The pre-test showed that respondents predominantly recalled events, domain by domain, and in chronological order. For example, respondents were more likely to relay their entire accommodation history from birth to present day than part of their work and residence history at the same time. By introducing a more standardised Blaise script and making the EHC more of a visual tool rather than the basis for the data entry screen, the program was more suitable for a large scale survey. The flow of questions was significantly improved and questions or checks could be triggered when gaps or overlaps were reported.

5.2.2 Moving forwards and backwards in time

It is fairly easy to move forwards and backwards in time using a paper EHC. However, complex programming is required to translate this into a CAPI interview. In the pre-tests and pilot, nearly all respondents said that they preferred forward chronological recall (for example beginning their work history from their first job and ending with their last or current occupation) and felt this was how they best remembered events. We therefore decided that incorporating programming functions which allowed backwards as well as

forwards recall into the CAPI program was an unnecessary complication, given that very few respondents remembered life events in this way. This also enabled us to improve the flow of the questions.

5.2.3 Order of topics

The flexibility of the paper based EHC for allowing respondents to choose and change the order in which they answer topics could easily be incorporated into the CAPI program using parallel blocks. The default order that topics are asked and appear on the EHC is: children and fertility, relationship history, accommodation, work history, health and other life events. In the CAPI pilot, respondents were explicitly asked to choose the order they wanted to answer topics in. However, at the beginning of the interview this is a difficult task for respondents to do. Many respondents were unsure which areas they would remember best and preferred to revert to the default order. In the main stage, respondents were no longer asked to choose the order, but interviewers were instructed that they can change the order if necessary at any time in the interview. This can be done using a simple key combination which takes the interviewer to the beginning of the required module. However, feedback suggests that most respondents were happy with the default order and therefore this function was very rarely used.

5.2.4 Revising and editing

With a paper calendar, revisions can be made using a pencil and eraser. For the CAPI interview, functions were developed which allowed interviewers to edit data previously entered when required. This was needed because the calendar can reveal inconsistencies which mean that the year an event occurred needs to be changed, or that the data relating to that event is incorrect. For example, a respondent may remember that they were a deputy head in 1990 but had previously recorded the job title as subject teacher. Some interviewers found this function easy to use and frequently used it during live interviews. However, this was quite technically challenging and therefore others did not use it so extensively or found it time consuming and difficult when they did.

Finding a way to insert an entire new event was most challenging (e.g. if the respondent realised in the interview that they had forgotten to say that they lived abroad for a year). Although, a function was created, this was too difficult to use in practice so unfortunately it was not part of the mainstage program. This may have affected data quality and the length of interviews. More development time and interviewer training could improve this and hopefully give a more workable and user-friendly solution.

6. Interviewer training

6.1 Interviewing skills

In a standardised Blaise interview, instructions, questions and probes are identical. The EHC method represents a shift in interviewing techniques and requires interviewers to generate personalised cues as the need arises. Furthermore, they are encouraged to check the information the respondent provides (using information displayed on the EHC such as respondent's age and personal events already entered). This needs to be done with subtlety and care to ensure the respondent does not feel they are being tested or undermined.

In addition to the change in interviewing style, the EHC presents a technical challenge to interviewers. The EHC utilises keys and key combinations which the interviewer would not use in a standard Blaise interview. For the less computer literate this was particularly difficult.

It was essential that interviewers were able to effectively use all the elements of the EHC and felt able to adapt to the (unconventional) style of interviewing the EHC approach entails. Training therefore focused on:

- (a) Familiarising interviewers with the EHC: workshops gave interviewers the opportunity to practice using the program, both at the briefing and for homework.
- (b) Interviewer style: screen capture software was used to give examples of respondent/interviewer interaction using the EHC.

6.2 Respondent and interviewer interaction with the EHC

An element of the EHC approach is that the respondent should interact with the EHC as well as the interviewer. Respondents should ideally be more involved in the process: a key aspect of this is being able to look at the calendar so that the respondent has the opportunity to see the visual prompts available (such as the personal events box) and possibly identify inconsistencies in their reports.

In practice, it was difficult for interviewers to adhere to this, especially when the CAPI EHC was used. Respondents did not always wish to sit next to the interviewer, and often the seating arrangement was not suitable for two people to sit and view the screen. The calendar was fairly difficult to see due to the size of the laptop screen, and this problem was heightened by both interviewers and respondents needing to 'share' the screen. Interviewers also found this approach difficult: they were unable to maintain eye contact with the respondent which they are used to in standardised CAPI interviewing, and disliked that respondents were able to read questions from the screen before they were asked by the interviewer.

Initial feedback from interviewers shows that most respondents did not look at the screen because they were not interested in the EHC. This problem may have been partly due to the age of the ELSA respondents (50 and over): bad eyesight or a general lack of experience with computers is more likely amongst this age group, whereas a different cohort may be more open to looking at the computer screen. Therefore, these early findings demonstrate that the calendar is a more effective tool for the interviewer than for the respondent. This shows how essential it is for interviewers to master the new interviewing technique (i.e. both the technical aspects of it and the more collaborative interviewing style) because responsibility for generating personalised cues and checking for inaccuracies falls to them.

7. Conclusion

The ELSA life history interview shows how an EHC can be incorporated into a standard Blaise style interview. This interview retained key features of the EHC approach: interviewers were able to generate personalised cues using the information displayed on the calendar and respondents could, to an extent, recall events in their own way. Thorough testing in the development stage showed that flexibility is not always necessary

for the EHC to be an effective tool to help respondents recall events. We are confident that despite the life history interview representing a more structured approach, the EHC we used enabled respondents to give more accurate and robust life histories. There are some areas which would benefit from more development time, particularly creating a more user-friendly way to insert new events and revise data already entered. This will be taken into account if we use the EHC for future projects. The feedback we have based this paper on is from respondent and interviewer de-brief reports and eighteen audio-recorded interviews in the development stage. We will be carrying out further analyses based on approximately 100 audio-recorded mainstage interviews. We will use behavior coding techniques to explore further the respondents' and interviewers' experiences of the ELSA life history interview and the effectiveness of the different elements of the calendar.

Navigating BCP with .NET

Peter Sparks, The University of Michigan

1. Abstract

The Blaise Component Pack is a gateway to expanding the capabilities of Blaise by working with metadata, data, and other parts of the Blaise system. However finding just the right information can be difficult, and the current documented examples are in Visual Basic 6 and C++. This paper looks at the different components available, how to navigate them from the various entry points using the .Net 2005 programming environment in both VB and C# and BCP 2.0, and some tips to make coding easier. Equivalent examples will be given in both Visual Basic.Net and C#.Net.

2. Introduction

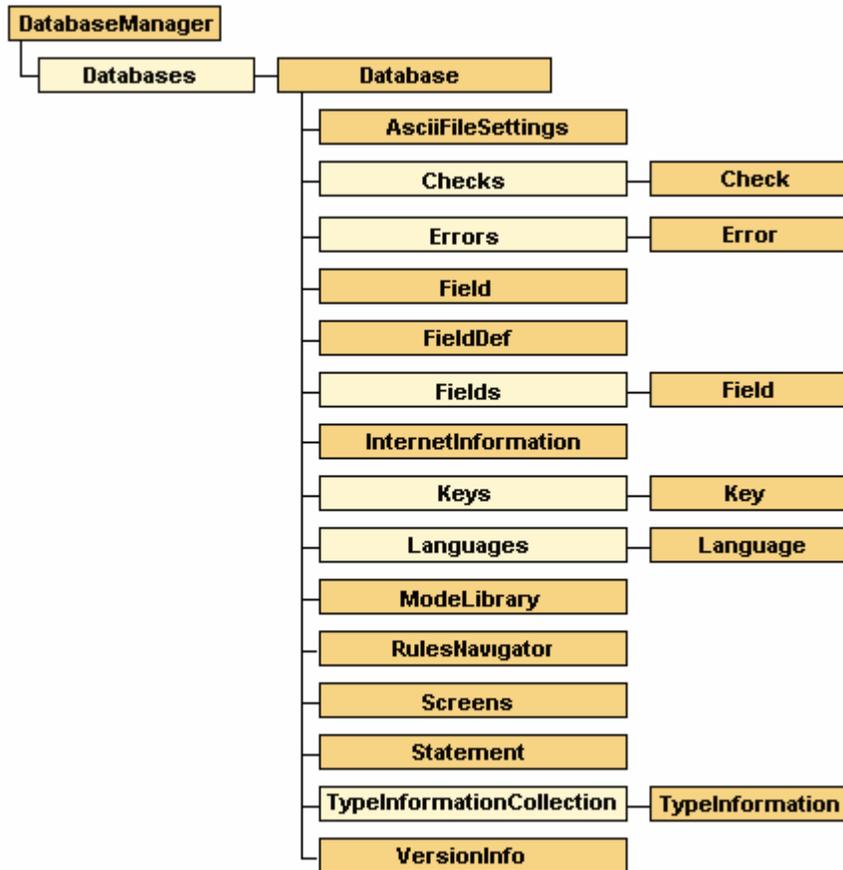
The BCP is a powerful tool for accessing information stored in the Blaise datamodel and database. It has been available for a number of years and has been used extensively by The University of Michigan in many of their utilities. As new programming environments have changed, the utilities have also been updated.

The change from Visual Basic 6 to C# in .Net 2.0 was not as straightforward as could be hoped. Some procedures and methods changed names, the way of retrieving indexed information in .Net was different, and the environment was changed. The documentation provided in the Blaise system uses VB6 and C++ and is a good starting point.

The purpose of this paper is to show several examples of common tasks to retrieve and modify information stored in the datamodel and database within the .Net environment. Appendices are provided to give more detail on certain topics, such as a VB6 to .Net translation guide, charts of method names between VB and C#, and some general programming tips.

This discussion uses the BCP (BLAPI4A2.dll) from Blaise 4.7 in this discussion. At the time of this writing a new Blaise 4.8 BLAPI3A.dll is now available.

3. Overview of the BCP Database Manager



The above illustration is from the Blaise 4.7 help “Blaise API Object Model.”

The Blaise Database Manager is the root object used to connect to the datamodel. As shown in the illustration there are many different entry points depending upon the task from the Database Manager. However, most of the functions needed are at the Database level.

The “Database” is somewhat misleading in that it is the way to access both the datamodel and the data stored with the datamodel.

There have been some additions and changes between VB6 and C#/VB.Net (see Appendix A). The syntax in using some of these methods has also changed.

One of the greater difficulties when first being exposed to the BCP is realizing that some commands are irrelevant if data is not being used. The discussion thus will start with strictly the datamodel, and then look at the data retrieved from the datamodel.

It is assumed the reader has familiarity with .Net peculiarities, such as being zero-based for its indexing. Appendix B has a brief discussion of the differences from VB6 to .Net.

The remarkable strength of working with the BCP is the ability to process and analyze data from the datamodel in a variety of ways. However, choose existing tools such as Cameleon when all that is needed is a quick list of variables or data dictionaries without needing a user interface. Create .Net programs when complex processing and interfaces are needed.

Although there are many possible tasks that can be accomplished using BCP only a few will be addressed in this paper.

3.1 Things you can do with BCP (datamodel)

1. Retrieve any field, auxfield, or local definition generally or specifically and associated information regardless where it is used
2. Step through all statements in the same order as defined in the rules, including blocks and procedures.
3. Retrieve any specific statement
4. Read through all the modelib settings
5. Read through the screen layouts defined for the DEP (main interview, parallel blocks)
6. Retrieve defined primary and secondary keys

3.2 Things you can do with BCP (database)

1. Read through all external lookups data files
2. Retrieve cases via specific keys, update fields, check case status, reevaluate the rules
3. Retrieve text with fills based upon the current case data

3.3 Other areas to explore (not in this paper)

1. Interact with the Data Entry Program (DEP) via Alien Routers and Alien Procedures
2. Retrieve Internet information
3. Navigate through the datamodel using data and the RulesNavigator
4. Explore all the type definitions, both system and user-defined
5. Process all errors in a form
6. Examine all checks and signals within the datamodel
7. Retrieve version information for a datamodel and a database
8. Make use of built-in database handling functions, such as copying or deleting Blaise databases.
9. Look at the language definitions for a datamodel

Given the lists above, it is surprising there are not some things you can do using the API. There are at least two areas that functionality has not been provided.

3.4 Things you cannot do with BCP

1. You cannot update or create datamodels. Blaise only allows the programmer to develop source code. The b4cpars.exe program can be used to prepare source code outside of the development environment.
2. You cannot work with compiled Manipula scripts, such as opening an .MSU and treat it as a datamodel.

4. Accessing BCP/API

4.1 Companion Source Code

A companion program and source code is available with this paper. All examples in this paper have been taken from the working program.

4.2 Initial steps – Accessing the datamodel

All work with the BCP begins first with Database Manager. Be sure you have made a reference to the BLAPI4A2.dll (Blaise 4.7) in your project references in order to work with BCP.

It is recommended that only one instance of the Database Manager is ever created. This is the entry point for both the Meta information and the data.

```
C#  
BlAPI4A2.DatabaseManager curDatabaseManager = new BlAPI4A2.DatabaseManager();
```

```
VB  
Dim curDatabaseManager As BlAPI4A2.DatabaseManager
```

In order to retrieve information from a datamodel and the associated database create a Database object.

```
C#  
BlAPI4A2.Database curDatabase;
```

```
VB  
Dim curDatabase As BlAPI4A2.Database
```

This defines the Database object but you have to use the OpenDatabase method of the DatabaseManager to get an instance. This still is not associated with either a datamodel or database.

```
C#  
curDatabase = curDatabaseManager.OpenDatabase("");
```

```
VB  
curDatabase = curDatabaseManager.OpenDatabase("")
```

To actually open the datamodel assign the full path and name of the datamodel to the DictionaryFileName property.

```
C#  
curDatabase.DictionaryFileName = @"c:\temp\SomeDatamodel.bmi";
```

```
VB.Net  
curDatabase.DictionaryFileName = "c:\temp\SomeDatamodel.bmi";
```

Now that the connection to the datamodel has been made, all meta information about the datamodel is available. This includes all fields, auxfields, locals, parameters, procedures, blocks, statements, screen layouts, modelib fonts, external datamodel definitions, categories, and more. The sole task is being able to get the information and process it.

4.2 The difference between Fields and Defined Fields

The main difference is the FieldDef object will give you the base information to the field as it is defined, while the Field object will give you an instance for every Field. For example, suppose there is a rostered field called Age. In Blaise it may be defined as

```
Age : ARRAY[1..20] OF 25..80
```

The FieldDef object will show this as one entity: Age[].

The Field object will give twenty entities: Age[1], Age[2], ..., Age[20].

There are additional methods and properties available only through the FieldDef object that are not in the Field object, such as the format of the external lookup files.

5. BCP/API Examples

Each of the following examples shows how to accomplish the task. The examples are a little shorter than the working programming code (removed procedure headings, Try/Catch statements, extra interface interactions) to keep the code short.

5.1 Retrieve any field, auxfield, or local definition generally or specifically and associated information regardless where it is used

There are two major ways to retrieve information needed from the datamodel: strictly all fields retrieved at one time, or recursively.

All at the same time has the advantage of simpler code but at the overhead cost of storing all those fields in memory at the same time.

So if working with datamodel with a large number of fields consider recursion, but at the cost of longer execution time.

There are properties that are specific to types of objects. In the Field/FieldDef examples below, the property .IndexedName has a non-null/non-Nothing value if the FieldKind property is not a generated parameter (BFieldKind.blfkGenParameter).

5.1.1 FieldDef: All fields non-recursive

Please note that using recursion strictly on a FieldDef will not work because the FieldDef does not have the DefinedFields()/ get_DefinedFields() method. If you need to use recursion and the FieldDef then use the Field recursive example below, retrieve the FieldDef for each field using the .FieldDef method, and finally store the results and remove duplicates.

get_DefinedFields returns a set of Fields and not FieldDef may be expected.

```
C#
BlAPI4A2.Fields curFields =
curDatabase.get_DefinedFields((int)BlAPI4A2.BlFieldKind.blfkAll,
(int)BlAPI4A2.BlFieldType.blftAll);

for (int i = 0; i < curFields.Count; i++)
{
    if (curFields[i + 1].FieldKind != BlAPI4A2.BlFieldKind.blfkGenParameter)
        lbResults.Items.Add(curFields[i + 1].FieldKind.ToString() + " "
+ curFields[i + 1].IndexedName);
    else
        lbResults.Items.Add(curFields[i + 1].FieldKind.ToString() + " "
+ curFields[i + 1].Parent.IndexedName + " GP");
}

VB
Dim curFields As BlAPI4A2.Fields =
curDatabase.DefinedFields(BlAPI4A2.BlFieldKind.blfkAll,
BlAPI4A2.BlFieldType.blftAll)

Dim i As Integer
For i = 0 To curFields.Count - 1
    If (curFields(i + 1).FieldKind <> BlAPI4A2.BlFieldKind.blfkGenParameter)
Then
        lbResults.Items.Add(curFields(i + 1).FieldKind.ToString() & " " _
& curFields(i + 1).IndexedName)
    Else
        lbResults.Items.Add(curFields(i + 1).FieldKind.ToString() & " " _
& curFields(i + 1).Parent.IndexedName & " GP")
    End If
Next i
```

5.1.2 Field: All fields Non-recursive

This routine is exactly like the non-recursive defined fields method with a different method, get_Fields(). It also returns a set of fields, but every field is made unique by the addition of the index number for sets and arrays. As a result many more fields are returned compared to the get_DefinedFields() method.

```
C#
BlAPI4A2.Fields curFields =
curDatabase.get_Fields((int)BlAPI4A2.BlFieldKind.blfkAll,
(int)BlAPI4A2.BlFieldType.blftAll);

for (int i = 0; i < curFields.Count; i++)
{
    if (curFields[i + 1].FieldKind != BlAPI4A2.BlFieldKind.blfkGenParameter)
        lbResults.Items.Add(curFields[i + 1].FieldKind.ToString() + " "
+ curFields[i + 1].IndexedName);
    else
        lbResults.Items.Add(curFields[i + 1].FieldKind.ToString() + " "
+ curFields[i + 1].Parent.IndexedName + " GP");
}

VB
Dim curFields As BlAPI4A2.Fields = curDatabase.Fields(BlAPI4A2.BlFieldKind.blfkAll,
BlAPI4A2.BlFieldType.blftAll)

Dim i As Integer
For i = 0 To curFields.Count - 1
```

```

Then      If (curFields(i + 1).FieldKind <> BlAPI4A2.BlFieldKind.blfkGenParameter)
          lbResults.Items.Add(curFields(i + 1).FieldKind.ToString() & " " _
          & curFields(i + 1).IndexedName)
Else
          lbResults.Items.Add(curFields(i + 1).FieldKind.ToString() & " " _
          & curFields(i + 1).Parent.IndexedName & " GP")
End If
Next i

```

5.1.3 Field: All fields Recursive

This uses the same method as before for retrieving all fields, but with a difference. The routine only recurses on those fields with a DataType of blftBlock. As a result each set of fields retrieved via the get_Fields()/Fields() method gets fields specific to that block, unlike the prior example where all fields of the datamodel were retrieved.

There is additional overhead and setup to work with recursion, and the general recommendation is to use it only if needed. It tends to have a greater toll on memory and processing time than other methods.

To start a recursive routine a starting value is always needed. In this example the root name of the datamodel is needed, and this is retrieved from the dictionary name curDatabase.DictionaryAsField.Name. The dictionary can be considered the uppermost block of the datamodel.

```

C#
int totfields = 0, lastPos = lbResults.Items.Count;
retrieveRecursive(curDatabase.get_Field(curDatabase.DictionaryAsField.Name), ref
totfields);

/// <summary>
/// Retrieve all fields in the datamodel using filters recursively
/// </summary>
/// <param name="curDatabase">Pass in the Blaise database object</param>
/// <param name="totfields">Variable to store the total number of
statements</param>
void retrieveRecursive(BlAPI4A2.Field curField, ref int totfields)
{
    if (curField.DataType == BlAPI4A2.BlFieldType.blftBlock)
    {
        BlAPI4A2.Fields curFields =
curField.get_Fields((int)BlAPI4A2.BlFieldKind.blfkAll,
                    (int)BlAPI4A2.BlFieldType.blftAll);

        totfields += curFields.Count;

        for (int i = 0; i < curFields.Count; i++)
        {
            if (curFields[i + 1].FieldKind != BlAPI4A2.BlFieldKind.blfkGenParameter)
                lbResults.Items.Add(curFields[i + 1].FieldKind.ToString() + " "
                + curFields[i + 1].IndexedName);
            else
                lbResults.Items.Add(curFields[i + 1].FieldKind.ToString() + " "
                + curFields[i + 1].Parent.IndexedName + " GP");

            if (curFields[i + 1].DataType == BlAPI4A2.BlFieldType.blftBlock)
                retrieveRecursive(curFields[i + 1], ref totfields);
        }
    }
}

VB
Dim totfields As Integer = 0, lastPos = lbResults.Items.Count
retrieveRecursive(curDatabase.Field(curDatabase.DictionaryAsField.Name), totfields)

''' <summary>
''' Retrieve all fields in the datamodel using filters recursively

```

```

''' </summary>
''' <param name="curField">The field to use as the starting point</param>
''' <param name="totfields">Variable to store the total number of
statements</param>
Private Sub retrieveRecursive(ByVal curField As BlAPI4A2.Field, ByRef totfields As
Integer)

    If (curField.DataType = BlAPI4A2.BlFieldType.blftBlock) Then
        Dim curFields As BlAPI4A2.Fields =
curField.Fields(BlAPI4A2.BlFieldKind.blfkAll,

                                BlAPI4A2.BlFieldType.blftAll)

        totfields += curFields.Count

        Dim i As Integer
        For i = 0 To curFields.Count - 1
            If (curFields(i + 1).FieldKind <>
BlAPI4A2.BlFieldKind.blfkGenParameter) Then
                lbResults.Items.Add(curFields(i + 1).FieldKind.ToString() +
" " -
                    + curFields(i + 1).IndexedName)
            Else
                lbResults.Items.Add(curFields(i + 1).FieldKind.ToString() +
" " -
                    + curFields(i + 1).Parent.IndexedName + " GP")
            End If

            If (curFields(i + 1).DataType = BlAPI4A2.BlFieldType.blftBlock)
Then
                retrieveRecursive(curFields(i + 1), totfields)
            End If
        End If
    Next i

End If
End Sub

```

5.1.4 Field: Filtered fields non-recursive

To get specific categories of fields change the filter parameters on the get_Fields()/Fields() method. For example, to get fields that store data in the database use the FieldKind with blfkDataField.

```

C#
BlAPI4A2.Fields curFields =
curDatabase.get_Fields((int)BlAPI4A2.BlFieldKind.blfkDataField,

                                (int)BlAPI4A2.BlFieldType.blftAll);

VB
Dim curFields As BlAPI4A2.Fields =
curDatabase.Fields(BlAPI4A2.BlFieldKind.blfkDataField,
                                BlAPI4A2.BlFieldType.blftAll)

```

The different FieldKind and FieldTypes are shown below. Note that C# requires type casting to (int) of the parameters for the method, whereas VB does this automatically.

blFieldKind	
blfkParameter	Field parameters are used within a block or procedure.
blfkGenParameter	The field is a generated parameter. This happens whenever a field reference from within a block is defined outside that block
blfkDataField	The field is stored in a Blaise database
blfkAuxField	The field is not stored in a Blaise database
blfkGenAuxField	Like a generated parameter, but based upon an Auxfield.
blfkExternal	A field reference to an external datamodel (lookup)
blfkLocal	Typically used for FOR loop indexes, counters, ...
blfkAll	Retrieve all field kinds

bFieldType	
bftUnknown	This should be an error condition if it occurs
bftString	String type
bftInteger	Numeric data as INTEGER or range x..y
bftFloat	Real data as REAL, REAL[n], REAL[n,m] or range x.d..y.c
bftDate	Datatype
bftTime	Timetype
bftMemo	Open type
bftEnumeration	Field contains categories
bftClassification	Classification type
bftExternal	The field is an external type
bftBlock	The field is a block (contains other subfields)
bftAll	All field types

5.1.5 Examining DK/RF, Sets, Arrays

Note that there are not filters available for other field attributes, such as whether the field may allow DK/RF, or whether the field is a set or an array. To look at DK/RF look at the `.DontKnowAllowed` and `.RefusalAllowed` properties on the field, and the `.Isxxx` properties on the `FieldDef` method. `curField` in the below example is defined as `BlAPI4A2.Field`.

```

curField.DontKnowAllowed
curField.RefusalAllowed

curField.FieldDef.IsArray
curField.FieldDef.IsEmbeddedBlock
curField.FieldDef.IsSet
curField.FieldDef.IsTable

```

5.2 Step through all statements in the same order as defined in the rules, including blocks and procedures.

A major requirement for many of the utilities used at The University of Michigan is to extract the field information in the same order of execution as the rules. This can be accomplished using the `RulesNavigator`, and can also be done via the `Statements` collection. Both methods use recursion.

The `RulesNavigator` is designed to be used with the data of a case and is documented within the Blaise help file. It makes use of a navigator pointed at particular statements, and can move back and forth throughout the instrument according to the data contained in the case. The `MoveToxxx` method is common to the navigator and causes the navigator pointer to change to the next appropriate statement. A true value is returned if the action was successful.

The `Statement` object is similar in concept to the `RulesNavigator`. It can be used with/without data, and also can follow the statements. The advantage of the `Statement` object is the ability to jump around the rules via the `StatementIdent`, whereas the `RulesNavigator` always starts at the first statement and then proceeds forwards from there.

The example below makes use of the Statement object. If the StatementIdent values are stored and indexed, then it is possible to process the rules of a datamodel in many different ways.

The routines uses of the .ThenStatement, .ElseStatement, and an implied .ChildStatement to navigate. The first statement of the datamodel is found at the DictionaryAsStatement property on the database. Only For loops, Blocks, and Procedures will have a statement collection associated with them that includes a count. All other types, such as assignment (Let), are considered as a standalone statement.

Navigating the datamodel via the rules can be like driving through a maze of cul-de-sacs (dead ends). The main roads are easy to navigate (block level), but there are subdivisions (sub blocks), houses (fields), and eventually the cul-de-sac at the end of the street. At the end of a block, the end of a procedure, the end of the IF statement, the Else statement, ... there will be no further statements. The method .NextStatement will return null/Nothing even though the next logical statement is just the next street over. You're not allowed to drive through the backyard to get to the next logical street. Hence, you have to back up to the last turn you made and go down the next street. This is the recursive element to navigating the datamodel.

The routine can be summarized as follows.

- Step through all statements in the current block (and process them as needed)
 - If the current statement groups other statements (loop, block, procedure), then step through all those substatements before continuing.
 - If the current statement is an IF, navigate the THEN portion, then navigate the ELSE portion.
- Once at the end of the current block, pop back up to the prior block where you last left off and continue processing.

```
C#
int totfields = 0;
retrieveRecursiveStatement(curDatabase.DictionaryAsStatement, ref totfields);

/// <summary>
/// Retrieve all statements including procedures
/// </summary>
/// <param name="curStatement">Starting statement</param>
/// <param name="totStatements">Variable to store the total number of
statements</param>
void retrieveRecursiveStatement(BlAPI4A2.Statement curStatement, ref int
totStatements)
{
totStatements += curStatement.Statements.Count;
if (curStatement.Statements.Count > 0)
for (int i = 1; i <= curStatement.Statements.Count; i++)
{
lbResults.Items.Add(curStatement.Statements[i].StatementType + "\t"
+ curStatement.Statements[i].StatementIdent + "\t"
+ curStatement.Statements[i].StatementText);

switch (curStatement.Statements[i].StatementType)
{
case (BlAPI4A2.BlStatementType.blstCondition):
retrieveRecursiveStatement(curStatement.Statements[i].ThenStatement,
ref totStatements);
if (curStatement.Statements[i].ElseStatement != null)
retrieveRecursiveStatement(
curStatement.Statements[i].ElseStatement,
ref totStatements);
```



```
C#
BlAPI4A2.Statement curStatement = curDatabase.get_Statement("12.35.7");
```

```
VB
Dim curStatement As BlAPI4A2.Statement = curDatabase.Statement("12.35.7")
```

5.4 Read through all the modelib settings

The modelib contains all the interface information that will be used with the DEP, such as screen layouts, screen ordering, contents of questions per page, custom and system fonts, and so forth. This is accessible via the modelib object.

Note that the information retrieved is via the modelib settings stored in the datamodel. This doesn't read the .BML file, nor does this read the configuration .DIW file.

5.4.1 Font information

The user-defined fonts are stored in the .ModeLibrary.Style.Fonts object. Use the get_CustomFont()/CustomFont() methods to retrieve information about the fonts. Only a few of the font options are shown below. The CustomFont expects a letter denoting the font being retrieved.

```
C#
string fontName = curDatabase.ModeLibrary.Style.Fonts.get_CustomFont("A").Name;
int fontSize = curDatabase.ModeLibrary.Style.Fonts.get_CustomFont("A").Size;
```

```
VB
Dim fontName As String = curDatabase.ModeLibrary.Style.Fonts.CustomFont("A").Name
Dim fontSize As Integer = curDatabase.ModeLibrary.Style.Fonts.CustomFont("A").Size
```

5.4.2 Read through the screen layouts defined for the DEP (main interview, parallel blocks)

The screen layouts are accessible via the Screens collection at the database level. They consist of Layout, Parallel, Page, and Quest collections and tend to work together.

5.4.2.1 Layout, Parallel, Page, and Quest collections

These four collections give access to when each field/auxfield is presented to the interviewer. By reading through these collections and comparing it against all fields/auxfields in the datamodel the defined but never-asked fields/auxfields can be located.

These collections represent a hierarchy, and can be shown below

Layout – such as Interviewing, Editing, or self-defined

 Parallel – such as the main parallel block, the primary key parallel, listing ...

 Page – each page contains a set of fields/auxfields that will appear on it

 Question – the actual field/auxfield can be referenced from here

```
C#
BlAPI4A2.LayoutSet Layout;
BlAPI4A2.Parallel Parallel;
BlAPI4A2.StoredPage Page;
BlAPI4A2.Question Quest;

int iLayout, iParallel, iPage, iQuest;
```

```

for (iLayout = 1; iLayout <= curDatabase.Screens.LayoutSetCollection.Count;
iLayout++)
{
    Layout = curDatabase.Screens.LayoutSetCollection[iLayout];
    for (iParallel = 1; iParallel <= Layout.ParallelCollection.Count; iParallel++)
    {
        Parallel = Layout.ParallelCollection[iParallel];
        for (iPage = 1; iPage <= Parallel.StoredPageCollection.Count; iPage++)
        {
            Page = Parallel.StoredPageCollection[iPage];
            for (iQuest = 1; iQuest <= Page.QuestionCollection.Count; iQuest++)
            {
                Quest = Page.QuestionCollection[iQuest];

                lbResults.Items.Add("Layout: " + iLayout.ToString()
                    + ", Parallel: " + iParallel.ToString()
                    + ", Page: " + iPage.ToString()
                    + ", Question: " + iQuest.ToString()
                    + ": " + Quest.Field.IndexedName);
            }
        }
    }
} /*for*/

VB
Dim Layout As BlAPI4A2.LayoutSet
Dim Parallel As BlAPI4A2.Parallel
Dim Page As BlAPI4A2.StoredPage
Dim Quest As BlAPI4A2.Question

Dim iLayout As Integer, iParallel As Integer, iPage As Integer, iQuest As Integer

For iLayout = 1 To curDatabase.Screens.LayoutSetCollection.Count
    Layout = curDatabase.Screens.LayoutSetCollection(iLayout)
    lbResults.Items.Add(vbTab & "Layout: " & Layout.Name)
    For iParallel = 1 To Layout.ParallelCollection.Count
        Parallel = Layout.ParallelCollection(iParallel)
        For iPage = 1 To Parallel.StoredPageCollection.Count
            Page = Parallel.StoredPageCollection(iPage)
            For iQuest = 1 To Page.QuestionCollection.Count
                Quest = Page.QuestionCollection(iQuest)
                lbResults.Items.Add("Layout: " & iLayout.ToString()
                    & ", Parallel: " & iParallel.ToString() _
                    & ", Page: " & iPage.ToString() _
                    & ", Question: " & iQuest.ToString() _
                    & ": " + Quest.Field.IndexedName)
            Next iQuest
        Next iPage
    Next iParallel
Next iLayout

```

5.5 Retrieve defined primary and secondary keys

Each key for a Blaise datamodel can consist of multiple parts. The number of keys via the .Keys property refer to both primary and secondary keys. To get the pieces that make up a key step through the InvolvedFields collection. The Keys.Kind method will be one of blkkPrimary, blkkSecondary, or blkkTrigram.

```

C#
for (int i = 0; i < curDatabase.Keys.Count; i++)
{
    string keyResult = "Name: " + curDatabase.Keys[i + 1].Name
        + ", Kind: " + curDatabase.Keys[i + 1].Kind + ", using ";

    for (int j = 0; j < curDatabase.Keys[i + 1].InvolvedFields.Count; j++)
    {
        if (j > 0)
            keyResult += ", ";

        keyResult += curDatabase.Keys[i + 1].InvolvedFields[j +
1].IndexedName;
    }

    lbResults.Items.Add(keyResult);
}

```

```

VB
Dim i As Integer
For i = 0 To curDatabase.Keys.Count - 1
    Dim keyResult As String = "Name: " & curDatabase.Keys(i + 1).Name & "
    & ", Kind: " & curDatabase.Keys(i + 1).Kind & "
using "

    Dim j As Integer
    For j = 0 To curDatabase.Keys(i + 1).InvolvedFields.Count - 1
        If (j > 0) Then
            keyResult += ", "
        End If

        keyResult += curDatabase.Keys(i + 1).InvolvedFields(j +
1).IndexedName
    Next j

    lbResults.Items.Add(keyResult)
Next i

```

5.6 Read through all external lookups data files

Reading an external datamodel is much like reading the main datamodel. The external references are found via the `BlAPI4A2.BlFieldKind.blfkExternal` type on the `get_DefinedFields()/DefinedFields()` method.

Once that is found, it is just a matter of stepping through each of those references, making a connection to the appropriate database, and then stepping through the fields and data contained therein.

The steps to read a data file are as follows.

1. Create a database connection via the `OpenDatabase()` method
2. For Ascii and AsciiRelational data files set the `AsciiFileSettings.Separator` and `.Delimiter` values
3. Assign the datamodel via the `DictionaryFileName`
4. Connect to the data via the `.Connect` property set to `True`
5. Use `.ReadNextRecord()` method to read the case data into memory
6. When finished, disconnect from the data files (`.Connect = False`), and dispose of the database connection.

The entire data record in text format is available via the `ActiveRecord.Value` property. Specific individual fields can be retrieved via `get_Field()/Field()` method, passing in full dot name of the field and looking at the `.Text` or `.Value` properties.

```

C#
BlAPI4A2.Fields oFields =
curDatabase.get_DefinedFields((int)BlAPI4A2.BlFieldKind.blfkExternal,
(int)BlAPI4A2.BlFieldType.blftAll);

for (int i = 0; i < oFields.Count; i++)
{
    BlAPI4A2.Database DBExt = curDatabaseManager.OpenDatabase("");

    if (oFields[i+1].FieldDef.ExternalInformation.StorageFormat ==
BlStorageFormat.blsfAscii ||
oFields[i+1].FieldDef.ExternalInformation.StorageFormat ==
BlStorageFormat.blsfAsciiRelational)
    {
        DBExt.StorageFormat = oFields[i +
1].FieldDef.ExternalInformation.StorageFormat;

```

```

        DBExt.AsciiFileSettings.Separator =
oFields[i+1].FieldDef.ExternalInformation.AsciiFileSeparator;
        DBExt.AsciiFileSettings.Delimiter =
oFields[i+1].FieldDef.ExternalInformation.AsciiFileDelimiter;
    }

    DBExt.DictionaryFileName =
oFields[i+1].FieldDef.ExternalInformation.DictionaryFileName;
    DBExt.DataFileName = oFields[i+1].FieldDef.ExternalInformation.DataFileName;

    // Make the connection to the Blaise database
    DBExt.Connected = true;

    string s = null;
    for (int j = 0; j < DBExt.ActiveRecord.Count; j++)
    {
        if (j > 0)
            s += ", ";
        s += DBExt.ActiveRecord[j + 1].Name;
    }

    lbResults.Items.Add("Database: " + DBExt.DataFileName + ", Columns = " + s);

    // Start reading through the records and display the results
    for (int j = 0; j < DBExt.RecordCount; j++)
    {
        DBExt.ReadNextRecord();
        s = null;
        for (int k = 0; k < DBExt.ActiveRecord.Count; k++)
        {
            if (k > 0)
                s += ", ";
            s += DBExt.ActiveRecord[k + 1].Value;
        }
        lbResults.Items.Add(s);
    }

    // close the database
    DBExt.Connected = false;

    // release the object for GC
    DBExt = null;

} /*for*/

VB
Dim oFields As BlAPI4A2.Fields =
curDatabase.DefinedFields(BlAPI4A2.BlFieldKind.blfkExternal,
BlAPI4A2.BlFieldType.blftAll)

Dim i As Integer
For i = 0 To oFields.Count - 1
    Dim DBExt As BlAPI4A2.Database = curDatabaseManager.OpenDatabase("")

    If (oFields(i + 1).FieldDef.ExternalInformation.StorageFormat =
        BlAPI4A2.BlStorageFormat.blsfAscii
    Or
    - oFields(i + 1).FieldDef.ExternalInformation.StorageFormat =
        BlAPI4A2.BlStorageFormat.blsfAsciiRelational)
    Then
        DBExt.StorageFormat =
oFields(i+1).FieldDef.ExternalInformation.StorageFormat
        DBExt.AsciiFileSettings.Separator =
oFields(i+1).FieldDef.ExternalInformation.AsciiFileSeparator
        DBExt.AsciiFileSettings.Delimiter =
oFields(i+1).FieldDef.ExternalInformation.AsciiFileDelimiter
    End If

        DBExt.DictionaryFileName =
oFields(i+1).FieldDef.ExternalInformation.DictionaryFileName
        DBExt.DataFileName = oFields(i+1).FieldDef.ExternalInformation.DataFileName

        ' Make the connection to the Blaise database
        DBExt.Connected = True

```

```

Dim s As String = Nothing
Dim j As Integer
For j = 0 To DBExt.ActiveRecord.Count - 1
    If (j > 0) Then
        s += ", "
    End If
    s += DBExt.ActiveRecord(j + 1).Name
Next j

lbResults.Items.Add("Database: " & DBExt.DataFileName & ", Columns = " + s)

' Start reading through the records and display the results
For j = 0 To DBExt.RecordCount - 1
    DBExt.ReadNextRecord()
    s = Nothing
    Dim k As Integer
    For k = 0 To DBExt.ActiveRecord.Count - 1
        If (k > 0) Then
            s += ", "
        End If
        s += DBExt.ActiveRecord(k + 1).Value
    Next k
    lbResults.Items.Add(s)
Next j

' close the database
DBExt.Connected = False

' release the object for GC
DBExt = Nothing

Next i

```

5.7 Retrieve cases via specific keys, update fields, check case status, and reevaluate the rules

A specific field for a case in a database can be updated, or the same field across all cases, and so forth can be updated using the BCP/API. The method is similar to that for external lookups. Make the connection to the database. Retrieve the specific case, then update the particular field. Write the results back to the database.

Note: the form status may change depending upon the mode that the update is being made. The mode, `DataEntryBehaviour`, can be changed to one of `bldbUncheckedEditing`, `bldbFreeInterviewing`, `bldbStrictInterviewing`, and `bldbCheckedEditing`.

The value for a field can only be changed via the `get_Field().Text/Field().Text` property. The `.Value` property is read-only and will be updated according to the value stored in the `.Text` property. The `.Text` property can accept either string representations of numeric values, string data appropriately formatted for the data type of the field, or category names for enumerated fields.

```

C#
curDatabase.DataFileName = databaseName;
curDatabase.Connected = true;

// Try to retrieve the specific case
curDatabase.KeyValue = key;
curDatabase.ReadRecord();

// guarantee that any field can be adjusted
curDatabase.DataEntryBehaviour =
BlAPIA2.BlDataEntryBehaviour.bldbUncheckedEditing;

// Note: the .Value is read-only and cannot be assigned

// and set the new value via the code name or the code value
curDatabase.get_Field(field).Text = data;

// cause any checks/signals to be noted

```

```

curDatabase.CheckRecord();

// save the results
curDatabase.UpdateRecord();

//
curDatabase.Connected = false;
curDatabase.DataFileName = null;

VB
curDatabase.DataFileName = databaseName
curDatabase.Connected = True

' Try to retrieve the specific case
curDatabase.KeyValue = key
curDatabase.ReadRecord()

' guarantee that any field can be adjusted
curDatabase.DataEntryBehaviour =
BlAPI4A2.BlDataEntryBehaviour.blDbUncheckedEditing

' Note: the .Value is read-only and cannot be assigned

' and set the new value via the code name or the code value
curDatabase.Field(field).Text = data

' cause any checks/signals to be noted
curDatabase.CheckRecord()

' save the results
curDatabase.UpdateRecord()

curDatabase.Connected = False
curDatabase.DataFileName = Nothing

```

5.8 Retrieve text with fills based upon the current case data

The datamodel has all the question text, but it is possible to retrieve the text with the fills per DEP if a dat has been associated with the datamodel. In this example assume the datamodel has been opened, and the database has been connected. The specific case to be retrieved is noted via the **key** variable.

The key point for this to work is make sure the BlDataEntryBehaviour has been set to interviewing. If it was left in data editing, the default mode, then the rules will not be executed and the fills will essentially come back as empty.

The example below steps through all the fields of the case and examines each question text. If it contains a '^' character signifying a fill it then gets the original text for the question and the filled text based upon the case data.

The first parameter of get_QuestionText()/QuestionText() is the language index, and the second is a boolean to indicate if fills should be replaced.

```

C#
curDatabase.DataEntryBehaviour =
BlAPI4A2.BlDataEntryBehaviour.blDbStrictInterviewing;

// Try to retrieve the specific case
curDatabase.KeyValue = key;
curDatabase.ReadRecord();

BlAPI4A2.Fields curFields =
curDatabase.get_Fields((int)BlAPI4A2.BlFieldKind.blfkDataField,

(int)BlAPI4A2.BlFieldType.blftAll);

for (int i = 1; i <= curFields.Count; i++)
    if (!string.IsNullOrEmpty(curFields[i].get_DefinedQuestionText(1)))
        if (curFields[i].get_DefinedQuestionText(1).Contains("^"))

```

```

        lbResults.Items.Add("Record #" + curDatabase.ActiveRecord.FormID + ",
Field="
                + curFields[i].IndexedName + ", Original Text="
                + curFields[i].get_QuestionText(1, false) + ", Filled
Text="
                + curFields[i].get_QuestionText(1, true));

curDatabase.Connected = false;
curDatabase.DataFileName = null;

VB
curDatabase.DataEntryBehaviour =
BlAPI4A2.BlDataEntryBehaviour.blDbStrictInterviewing

' Try to retrieve the specific case
curDatabase.KeyValue = key
curDatabase.ReadRecord()

Dim curFields As BlAPI4A2.Fields =
curDatabase.Fields(BlAPI4A2.BlFieldKind.blfkDataField,
BlAPI4A2.BlFieldType.blftAll)

Dim i As Integer
For i = 1 To curFields.Count
    If Not (String.IsNullOrEmpty(curFields(i).DefinedQuestionText(1))) Then
        If (curFields(i).DefinedQuestionText(1).Contains("^")) Then
            lbResults.Items.Add("Record #" & curDatabase.ActiveRecord.FormID & ",
Field=" _
                    & curFields(i).IndexedName & ", Original Text=" _
                    & curFields(i).QuestionText(1, False) & ", Filled
Text=" _
                    & curFields(i).QuestionText(1, True))
            End If
        End If
    End If
Next i

curDatabase.Connected = False
curDatabase.DataFileName = Nothing

```

6. Conclusion

As shown in this paper, the information available via the BCP/API is rich in detail and can be used to accomplish many tasks. This Blaise “engine” makes it possible to create many utilities that are beyond the capabilities of Manipula, Maniplus, and Cameleon. Although the current Blaise documentation (4.7) is based upon VB6, there is a direct translation to the .Net environment. This is important since the development tools are always changing and VB6 is no longer supported by the Microsoft Corporation.

7. Appendix A

Below are tables comparing the use of VB commands in the documentation with the C# equivalent. An item in *italics* signifies it was not part of the documentation in VB6.

Database

Data?	External?	Database: VB	Database: C#
Yes		AccessMode	AccessMode
Yes		ActiveKey	ActiveKey
Yes		<i>ActiveRecord</i>	<i>ActiveRecord</i>
	Yes	AsciiFileSettings	AsciiFileSettings
		BlocksWithData	BlocksWithData
Yes		BOF	BOF
		CATI	<i>Cati</i>
		CAWI	<i>Cawi</i>
		Checks	Checks
Yes		Connected	Connected
Yes		CreateOnOpen	CreateOnOpen
Yes		DataChangedByRules	DataChangedByRules
Yes		DataChangedByUser	DataChangedByUser
Yes		DataEntryBehaviour	DataEntryBehaviour
Yes		DataFileName	DataFileName
		DataFilesInUse	DataFilesInUse
		DataFileVersionInfo	DataFileVersionInfo
		DictionaryAsField	DictionaryAsField
		DictionaryAsStatement	DictionaryAsStatement
		DictionaryChecksum	DictionaryChecksum
		DictionaryCreationDate	DictionaryCreationDate
		DictionaryCreationTime	DictionaryCreationTime
		DictionaryFileName	DictionaryFileName
		DictionaryFileVersionInfo	DictionaryFileVersionInfo
		DictionaryVersion	DictionaryVersion
Yes		EOF	EOF
	Yes	ExternalSearchPath	ExternalSearchPath
		FieldFilter	FieldFilter
Yes		FormID	FormID
Yes		FormStatus	FormStatus
		IMGLink	IMGLink
		InOutMode	InOutMode
		Internet	Internet
		InternetInformation	InternetInformation
		Keys	Keys
		KeyValue	KeyValue
		Languages	Languages
Yes		LastRecord	LastRecord
		Libraries	Libraries
		ModeLibrary	ModeLibrary
		ParallelDefs	ParallelDefs
Yes		PrimaryKeyFieldValues	PrimaryKeyFieldValues
Yes		RecordCount	RecordCount
Yes		RecordFilter	RecordFilter

Data?	External?	Database: VB	Database: C#
		RulesNavigator	RulesNavigator
		Screens	Screens
Yes		SmallErrorStack	SmallErrorStack
Yes		StorageFormat	StorageFormat
		SystemTypes	SystemTypes
		UsedDictionaries	UsedDictionaries
		UserTypes	UserTypes

Database

For Data?	Database methods: VB	Database methods:C#
Yes	CheckRecord()	CheckRecord()
Yes	ClearRecord()	ClearRecord()
Yes	CreateRecord()	CreateRecord()
Yes	DeleteRecord()	DeleteRecord()
Yes	ReadRecord()	ReadRecord()
Yes	ReadNextRecord()	ReadNextRecord()
Yes	ReadPreviousRecord()	ReadPreviousRecord()
Yes	<i>ReadRelativeRecord(OffSet)</i>	<i>ReadRelativeRecord(OffSet)</i>
Yes	<i>ReadRelativeRecords(OffSet, Count, BlaiseRecord)</i>	<i>ReadRelativeRecords(OffSet, Count, BlaiseRecord)</i>
Yes	Reset()	Reset()
Yes	UpdateRecord()	UpdateRecord()
Yes	WriteRecord()	WriteRecord()
	<i>DefinedFields(BIFieldKind, BIFieldType)</i>	<i>get_DefinedFields(BIFieldKind, BIFieldType)</i>
	<i>DictionaryText(Language)</i>	<i>get_DictionaryText(Language)</i>
Yes	<i>Errors(BLErrorKind)</i>	<i>get_Errors(BLErrorKind)</i>
	<i>Fields(BIFieldKind, BIFieldType)</i>	<i>get_Fields(BIFieldKind, BIFieldType)</i>
Yes	<i>NextFieldOnRoute(Field, AConditionSet)</i>	<i>get_NextFieldOnRoute(Field, AConditionSet)</i>
Yes	NextPageOnRoute(Question, Toggles)	NextPageOnRoute(Question, Toggles)
Yes	NextQuestionOnRoute(Question, Toggles)	NextQuestionOnRoute(Question, Toggles)
	<i>Field(Name)</i>	<i>get_Field(Name)</i>
	<i>FieldDef(Name)</i>	<i>get_FieldDef(Name)</i>
Yes	FirstQuestionOnRoute(Layout, Parallel, Toggles)	FirstQuestionOnRoute(Layout, Parallel, Toggles)
Yes	<i>PreviousFieldOnRoute(Field, AConditionSet)</i>	<i>get_PreviousFieldOnRoute(Field, AConditionSet)</i>
Yes	PreviousPageOnRoute(Question, Toggles)	PreviousPageOnRoute(Question, Toggles)
Yes	PreviousQuestionOnRoute(Question, Toggles)	PreviousQuestionOnRoute(Question, Toggles)
	QuestionAllowed(Question, Toggles)	QuestionAllowed(Question, Toggles)
	<i>Statement(Ident)</i>	<i>get_Statement(Ident)</i>

Field

Data?	Field methods: VB	Field methods:C#
	ArrayIndex	ArrayIndex
	Attributes	Attributes
	CategoryTexts(Language)	get_CategoryTexts(Language)
	ColumnTitles	ColumnTitles
	Database	Database
	DataType	DataType

Data?	Field methods: VB	Field methods:C#
	DefinedDescriptionText(Language)	get_DefinedDescriptionText(Language)
	DefinedFields(BIFieldKind, BIFieldType)	get_DefinedFields(BIFieldKind, BIFieldType)
	DefinedQuestionText(Language)	get_DefinedQuestionText(Language)
	DefinedRowTitles	DefinedRowTitles
	DefinedTag	DefinedTag
	DescriptionText(Language, Substitute)	get_DescriptionText(Language, Substitute)
	DisplayText	DisplayText
	DontKnowAllowed	DontKnowAllowed
	Editable	Editable
	EditInformation	EditInformation
	EditMode	EditMode
	EnumerationDisplayString	get_EnumerationDisplayString(ShowLabels)
	ErrorCounter(BIErrorKind)	get_ErrorCounter(BIErrorKind)
	Errors(ErrorKinds)	get_Errors(ErrorKinds)
	FieldDef	FieldDef
	FieldKind	FieldKind
Y	Fields(BIFieldKind, BIFieldType)	get_Fields(BIFieldKind, BIFieldType)
	IndexedName	IndexedName
	IsKeyField(BIKeyKind)	get_IsKeyField(BIKeyKind)
	IsParallel	IsParallel
	LocalName	LocalName
	Name	Name
	Parent	Parent
Y	QuestionText(Language, Substitute)	get_QuestionText(Language, Substitute)
	ReferredFieldName	ReferredFieldName
	RefusalAllowed	RefusalAllowed
	Remarkd	Remarkd
	RemarkText	RemarkText
	Required	Required
	RouteStatus	RouteStatus
	SetDisplayString	SetDisplayString
	Size	Size
	Status	Status
	Tag	Tag
Y	Text	Text
	TextAsSet	TextAsSet
Y	Value	Value
	AheadOnRoute(Field)	AheadOnRoute(Field)
	CopyDitto()	CopyDitto()
	Validate	Validate

FieldDef

FieldDef methods: VB	FieldDef methods:C#
BlockText(Language)	get_BlockText(Language)
Categories	Categories
Classification	Classification
Data Type	Data Type
Decimals	Decimals
DisplayDataType	DisplayDataType
DisplayElementDataType	DisplayElementDataType

ExternalInformation	ExternalInformation
IsArray	IsArray
IsEmbeddedBlock	IsEmbeddedBlock
IsSet	IsSet
IsTable	IsTable
LocalName	LocalName
MaxIndex	MaxIndex
MaxValue	MaxValue
MinIndex	MinIndex
MinValue	MinValue
Name	Name
RulesText (WithGeneratedParams, WithGeneratedRules)	get_RulesText(WithGeneratedParams, WithGeneratedRules)
Size	Size
Statements	Statements
TypeInformation	TypeInformation
CatNameToInt (Value)	CatNameToInt(Value)
GetDisplayText (FieldValue)	GetDisplayText(FieldValue)
IntToCatName(Value_)	IntToCatName(Value_)
ValidateClassificationCode(Value_)	ValidateClassificationCode(Value_)

Categories

Field methods: VB6	Field methods:C#/VB.NET
FieldDef.Categories.Count	FieldDef.Categories.Count
FieldDef.Categories.IndexOf(Name)	FieldDef.Categories.IndexOf(Name)
FieldDef.Categories[].Code	FieldDef.Categories[].Code
FieldDef.Categories[].Name	FieldDef.Categories[].Name
FieldDef.Categories[].Text(Language)	FieldDef.Categories[].get_Text(Language)
FieldDef.Categories[].TextDefined(Language)	FieldDef.Categories[].get_TextDefined(Language)

8. Appendix B

Differences between VB6 and .Net

For programmers familiar with VB6 and not so much .Net there are a few pitfalls that can be avoided with some care.

- **Indexing**
VB6 has a base index of “1” and so does the BCP, such as `Fields[1].IndexedName`. .Net works with a base index of “0,” so stepping through an indexed variable is slightly different.

```
VB6
Set oFields = _Database.DefinedFields(blfkAll, blftAll)
For I = 1 To oFields.Count
    curName(I) = oFields(I).IndexedName
Next I
```

```
VB.Net
oFields = _Database.DefinedFields(BlFieldKind.blfkAll,
                                   BlFieldType.blftAll)

For I = 0 To oFields.Count - 1
    curName(I) = oFields(I - 1).IndexedName
Next I
```

```
C#.Net
oFields = _Database.get_DefinedFields((int)BlFieldKind.blfkAll,
                                       (int)BlFieldType.blftAll);

for (int I = 0; I < oFields.Count; I++)
{
    curName[I] = oFields[I - 1].IndexedName;
}
```

- **Case sensitivity**
.Net is case-sensitive. This means that “count” is not the same variable as “Count.” The compiler will let you know.
- **Strong typing**
Assignments and parameters must have the same type. VB6 allowed free assignment between strings and integers and behind the scenes did the conversion. This sometimes lead to unexpected results.
- **Defined variables**
All variables used in .Net must be declared, and initialized depending upon their use.
- **Object orientation**
.Net is strongly object-oriented. It has a large impact on how memory and processing time works in the programs written. Follow good programming practices and declare and use objects locally, dispose or assign them to null/Nothing when finished, avoid heavy string use but use string builders, and use the `IDisposable` inheritance on your own classes.

9. Recommendations for working with .Net

Error trapping

It is highly recommended that you use Try/Catch/Finally for each procedure you write and do your best to handle errors. Use a simple log file and write errors to the log, especially for complex routines. This will greatly reduce the time and effort of making your program robust.

Debugging log

By the same regard, create a debugging log that is written when a debugging flag has been set.

.Net Development Environment

This is a very powerful tool for developing your applications, and again a great deal of time can be saved by learning as many features of the Visual Studio environment as possible.

Basic BCP/API recommendations

Retrieve collections all at the same time if you can afford the memory use, and then step through the collection using an integer index. This will allow you to quickly step through all the information. However, be warned: retrieving large quantities of data at one time makes a hit on your system resources.

Although the BCP/API does a good job of releasing objects, it behaves like strings in the .Net environment. That is, the objects are collected eventually when garbage collection happens. You can force garbage collection earlier by the GC() method. For this reason limit creation of new objects to those actually needed, keep the object definition as local as possible, avoid global objects that are held around a long time, and create classes that use the Dispose() inheritance.

10. Appendix C

10.1 Additional .Net tips

Shorthand for if/else

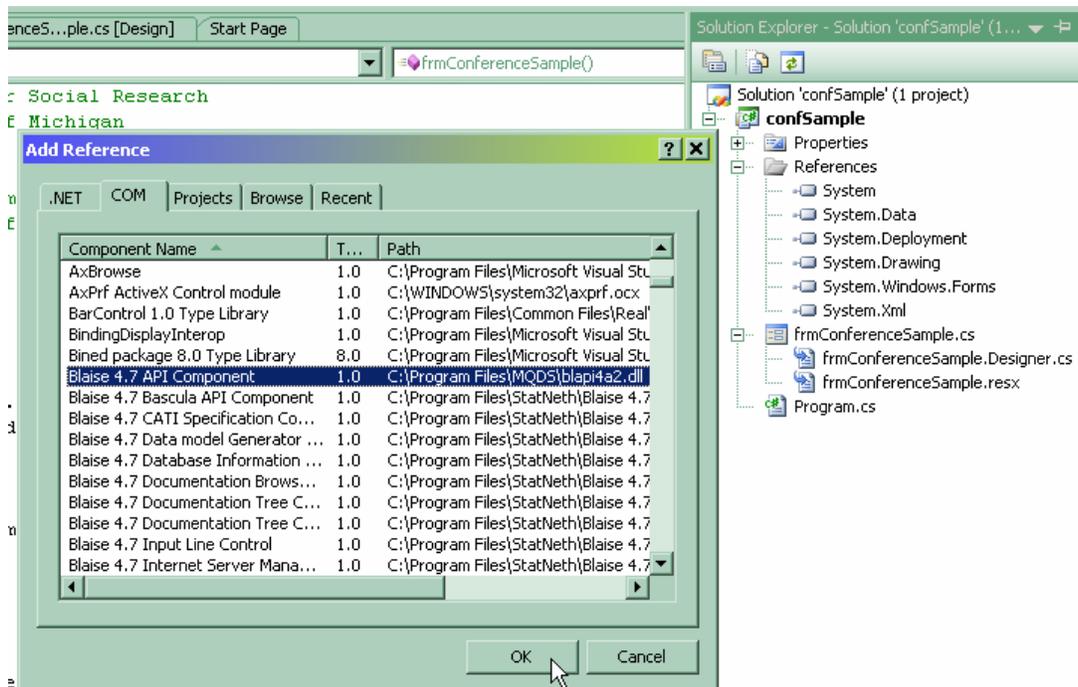
```
curDatabaseManager.DictionarySearchPath = pos > -1 ? txtBMIFile.Text.Substring(0, pos) : "\\\";
```

Escaping string sequences

Use the backslash for each escape, such as ‘\’ to get the character ‘\’, or the @ sign in front of a string to escape all the dereferences, such as @“c:\program files\thisandthat\sample.txt.”

Adding a BCP/API reference to your .Net solution.

- Create a solution.
- Next, open the Solution Explorer and open the References tree. Right-click and Add a Reference. Choose the “COM” tab, and select the “Blaise 4.7 API Component” item. This will add both the BLAPI4A2.dll and the SiStrCIA.dll to your project.
- If you don’t like typing the BLAPI4A2 class name repeatedly you can make a reference at the top of your C# program with the using statement, or the imports statement in VB.



Lifecycle Processes to Insure Quality of Blaise Interview Data

Linda Gowen and Pat Clark, Westat, Inc.

1. Introduction

Efficient, accurate and timely processing of interview data post-collection is critical in both cross sectional and longitudinal survey environments. Through the use of the Blaise API a comprehensive approach has been implemented for the processing of Blaise interview data through the entire survey lifecycle. The Blaise API makes it possible to use 'add-on' tools and processes at each of the survey lifecycle stages to insure data quality. This paper reviews some of the tools used at Westat with a focus on the dynamics of managing processes impacting data quality. The experiences implementing a lifecycle approach for COTS Blaise studies across multiple survey environments within an organization are discussed.

2. Survey Lifecycle Overview

Tools have been developed to support each of the major lifecycle stages in a COTS Blaise study including design, development, data collection and post-collection processing, and data delivery. At the start of each study, a core project team with the major stakeholders (Client, Project Director, Systems Manager, and Data Manager) decides which tools to utilize in conjunction with the particular study. The major lifecycle stages are as follows:

I. Design

- Requirements Analysis
- Develop CAI Specification

II. Development

- Develop Blaise System
- Internally Test Blaise System.
- Pretest for Field Readiness

III. Data Collection

- Deploy Blaise System

IV. Backend/Data Preparation

- Edit and QA Blaise Data
- Create Codebooks

V. Data Delivery

- Deliver Data
- Data Documentation

Figure 1 shows the major tasks in each of the stages and the dynamic interplay between stages throughout the lifecycle. Blaise provides many system capabilities used to integrate external tools and processes including:

- Blaise API – permits use of relational databases and data manipulation through other programming languages such as Visual Basic
- Manipula - batch system to import, export, and recode data that can be used for reporting and data manipulation
- Lookup files can be incorporated into the design as separate databases and can be searched to prevent misspellings and to facilitate text entry during interview
- External data files can be in ASCII format

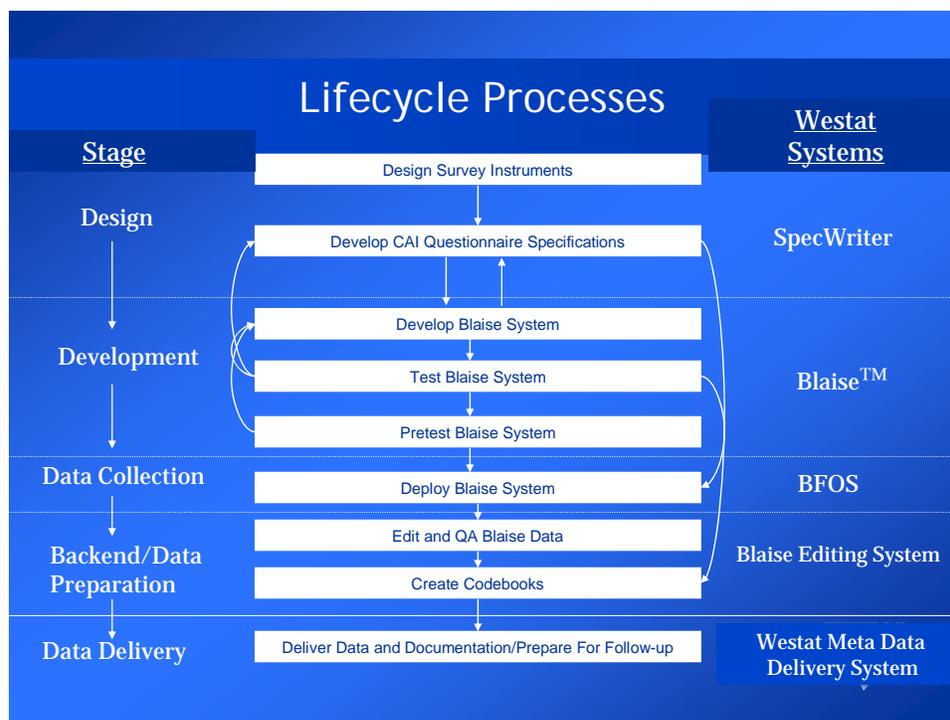


Figure 1: Lifecycle Processes

2.1 Stage 1 – Requirements Analysis

The goal of this stage is to define both the Blaise and non-Blaise requirements for the system. Since one Blaise system is never a stand alone system, it is necessary for the Systems Manager to prepare a project data flow diagram. It must show all sources providing input data to the all Blaise Systems, as well as, the data required to feed external sources. The details of external system integration should be designed and incorporated into the CAI specifications. The data flow diagram will provide a detailed illustration of how all data are input and output by the Blaise system. Simply put, the flowchart should illustrate how the individual questionnaire data will

move through all systems needed for processing and how the different systems work together.

2.2 Stage 2 – Develop CAI Specifications

In this stage the CAI specifications document will be created. This document will include the “complete” hard copy questionnaire typically needed for regulatory compliance with additional detailed instructions necessary for programming the CAI system.

Walkthroughs with the entire team are an important component of the specification development process. All perspectives need to be considered including the research objectives and costs of various approaches. In the paper-and-pencil environment, special cases were resolved by an interviewer writing marginal notes. In CAI, these situations need to be anticipated in the specifications if special question sequences need to be programmed. In CAI interviewers can always enter comments, but like marginal notes these comments require special review post-collection in order to assure data quality. These types of considerations should be addressed in the CAI specification development process.

We use a tool called SpecWriter, to help capture the hard copy content and programming specifications for the instrument. It is a front-end application built to be a survey development tool as well as capture details needed for programming. Its purpose is to integrate the project design and systems development of survey questionnaires. The tool is comprehensive and provides the ability for all team members to make their contributions in one location. The study staff can design and specify survey question-and-answer sequences, logic checks, and question routing, while data management and systems staff can use it to add detailed logics; edit formulas and information about the Blaise structure.

SpecWriter facilitates versioning by tracking changes to the instrument. Changes are made easily in the system all the while documenting and revising specifications. Each definitive version of the instrument can be identified with a version number. The version number assigned to the instrument, flow-chart and hard-copy questionnaire should be consistent so that it is clear what relationship each has to the other. One can also use version numbers to control and track changes, maintaining a log listing the changes made to each. Logging with versioning is useful in the development and maintenance of the CAI instrument because it provides a history of decisions along with resulting changes. This helps to avoid errors that may be repeated given the complexities in the particulars of a project.

SpecWriter Output

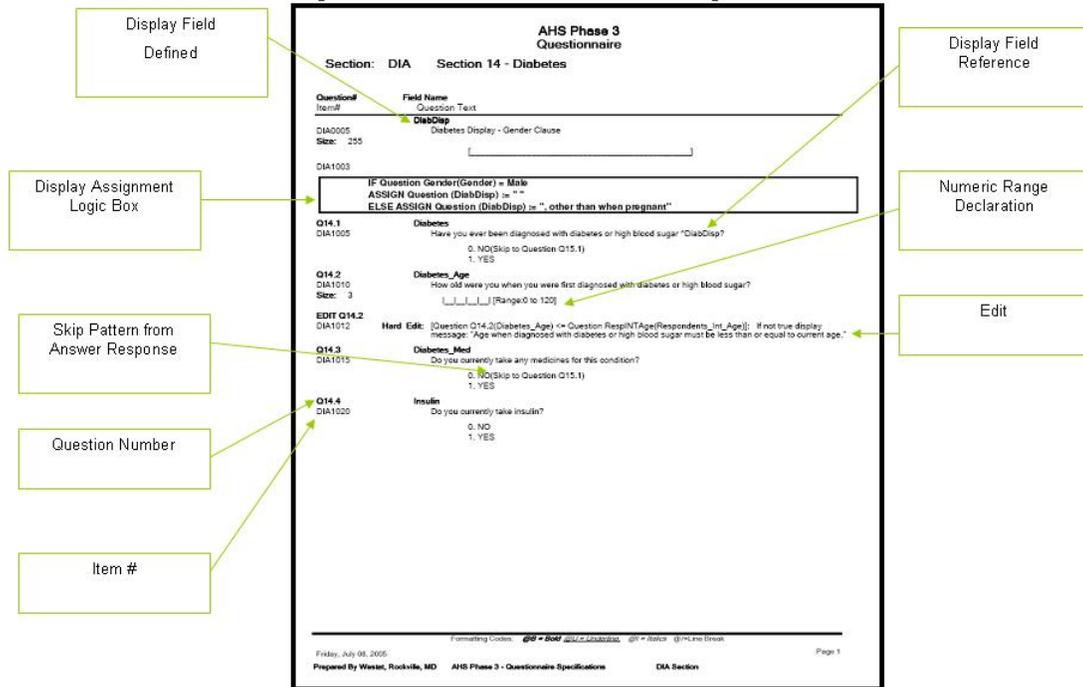


Figure 2: SpecWriter Report

SpecWriter creates several version controlled reports with varying levels of detail for use by: clients, project managers, data managers, and programmers. The ability to present specifications at varying levels of specificity helps to bridge the language gap between programmers and study managers. Another benefit of using SpecWriter is that it can create programming code for inclusion into the Blaise system.

2.3 Stage 3 – Blaise Development

Programmers use the output from SpecWriter to begin source code development, and the data flow diagram in the assembly of all components of the overall system. An important consideration for Blaise system development is to know how Blaise stores the data in conjunction with the field and block declarations. Also, it is essential to know how data are extracted out of the system. Testing should include extraction procedures by checking the creation of output files and variables. Navigation and repeating blocks need careful review to avoid data overwriting situations. It may be necessary for programming to include passing data between blocks for explicit creation of foreign keys.

Key considerations to assure that the collected data can be properly transformed for data delivery at the end of the data collection period include:

1. Proper use of data to be used as foreign keys in the creation of blocks/tables as the instrument is coded
2. Review of sampling or randomization processes being coded
3. Testing and QC at all stages of instrument development including verification of the data integrity after both forward and backward navigation
4. Designation of field name lengths appropriately for later SAS processing;
5. Relationship of question numbers to Blaise variable names, and
6. Ensuring that all needed intermediate variables are saved

During production and post-processing, Blaise data may be needed by other processing systems. Advance planning for reporting is considered so the programming staff can assure the database structure provides for the easy extraction of the needed information, or that programs are developed to permit viewing within the Blaise database itself. Timing variables and status variables should be included in the code to support reporting. These requirements should be captured in the CAI specifications document.

2.4 Stage 4 - Internally Testing the Blaise System

A specialized testing team ensures a level of quality in systems and data that has become more difficult to attain as the infrastructure becomes more complex. Testing is done using all systems that will be functioning in the field or in the home office. Using the business requirements, questionnaire specifications, and specifically designed 'test' cases, allows for thorough testing of each instrument. All testing is based on workflow with regard to how users will do their jobs, based on all the provided requirements and specifications. Functional testing is the main focus, which is to verify that each system conforms to all requirements as stated in the specifications. Feature by feature validation is performed, making sure that all skip patterns work properly, and using boundary values and erroneous input data to check error messages. System documentation such as user guides and training materials are tested to ensure accuracy.

As with any software development effort, it is important to track problems and their resolutions when developing a Blaise instrument. This ensures that all problems have been noted, and either resolved or had a decision made in regards to it. Track Integrity, an off-the-shelf product, provides robust capability for tracking problems including numerous customizations features. Users are able to configure the system to meet specific requirements in their problem tracking and resolution process, including the option to add user-defined variables.

As tests are run, new requirements may emerge based on issues reported in testing. The schedule may be adjusted accordingly, test cases are updated, and decisions on

all issues are recorded in the issue tracker. Final testing approval is measured by successful completion of all test cases and by the closeout of all issues. Although testing approval may have been received on the fielded instruments, any subsequent changes made to those instruments require additional "regression" testing. Regression testing includes the testing of all current valid test cases as well as new ones resulting from the new requirements or fixes. This type of testing is essential to quality assurance because it eliminates the possibility of introducing unexpected errors to previously working functions. Regression testing, particularly on a longitudinal study, is an integral part of quality assurance throughout the lifetime of the study.

2.5 Stage 5 - Pretest for Field Readiness

While specialized testing is an important aspect of an instrument's lifecycle, it is important that testing is also performed by the project, data management staff, and field staff. It is vital to identify idiosyncrasies that will need to be dealt with in training or in post-collection activities.

A good place to start is in constructing the mock data for the role plays that will be used in training. These role plays should simulate data collection scenarios. The goal is to have each condition tested for all the responses available in the instrument, to check that the appropriate paths are followed. An automated training script system, WinCATS, has been used in designing role plays. It creates screen captures of the Blaise instrument and annotates the document with imported graphic images designed to replicate the question/response flow of an interview. The electronic format allows for easy updating when changes are made to the collection instrument. Once the mock data has been constructed, the full cycle of the system is tested from input through delivery. Not only does this give confidence in the functioning of the questions, but it allows the team to pinpoint questions where specialized coding will be needed and areas that might need emphasis during training. Files with the delivery layout are built from the mock data and frequencies run to verify that the system is capturing the data correctly.

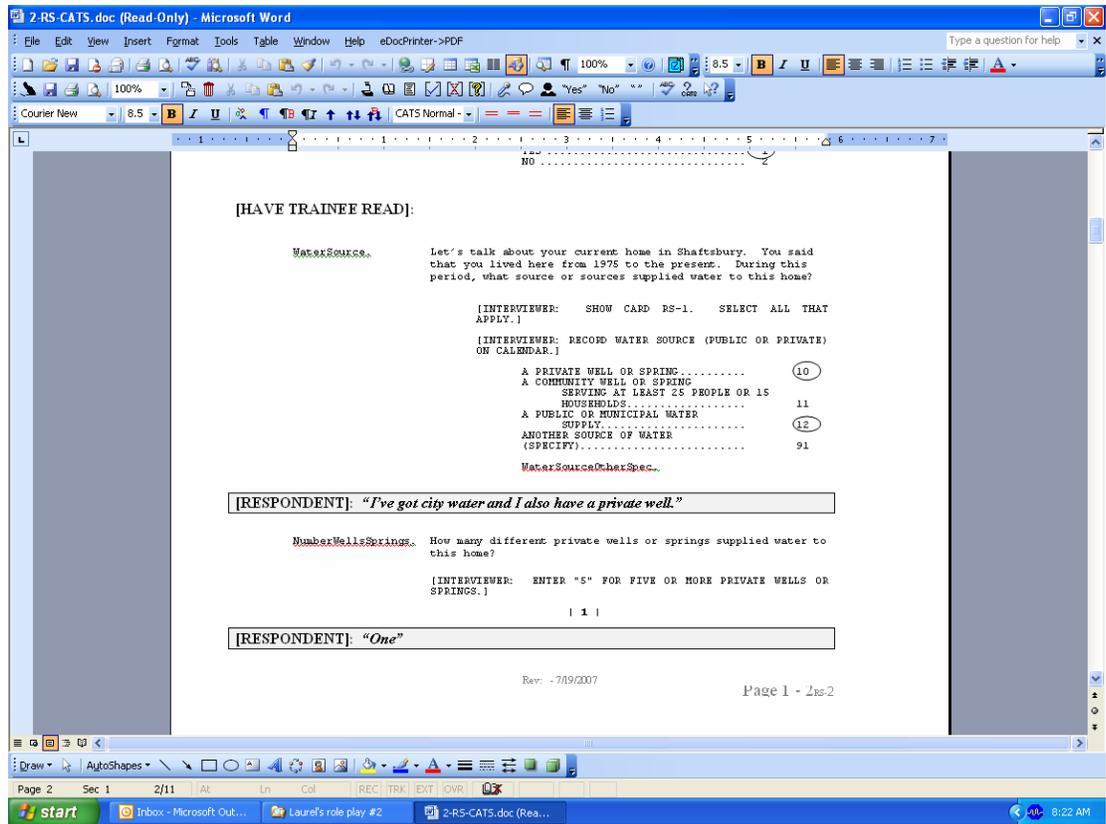


Figure 3: Example of WINCATS role play document.

2.6 Stage 6 – Blaise System Deployment

Deploying Blaise systems to the field can vary in complexity depending on the suite of programs needed for the management and distribution of the Blaise software and its accompanying data. Although projects vary in their implementation, all projects have field management or telephone center management systems supporting the Blaise applications. Westat's Basic Field Operations System (BFOS) provides CAI projects with a baseline field management system designed around a field model common to most Westat field data collection projects. In addition, BFOS can be adapted and customized to meet more specialized project needs.

More specifically, BFOS provides projects with the following:

- **Interviewer Management System** - A laptop-based application that allows interviewers to review and change assigned case status, launches the Blaise CAPI instrument, and synchronizes management and CAPI data with the home office.

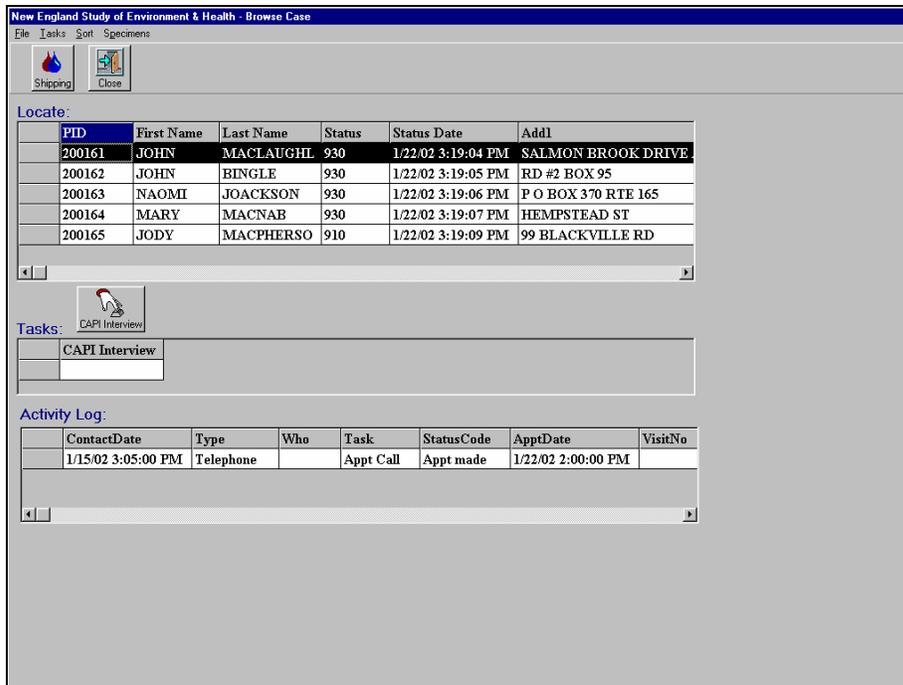


Figure 4: BFOS Case Browser

- **Supervisor Management System** - This is the web-based application used by field supervisors to view and manage the work of interviewers, including case assignment and transfer between interviewers, review and adjustment of case status, and other administrative and reporting functions.
- **Home Office Management System** - This provides functions needed by home office staff to load and manage the study sample information and to maintain information about the field staff and their assignments.

A very important aspect to the quality of a Blaise system is to closely track when the different versions of a Blaise application are deployed. Each new data model should always include a version field that is changed to reflect the field release number. This helps with troubleshooting and can be used for documenting in a codebook. The version can be used in codebook inapplicable statements that document why a field is not answered. The version field can be used to identify when a field was introduced in a subsequent release. The practice of setting the version field to the new value will force a new version of the Blaise data model. This is good because it forces a conversion from the previous model to the new model which will force the project to address differences in the data early in the process. Also the deployment process should include logging of dates and times when particular versions were implemented.

2.7 Stage 7 - Editing Blaise Data

A Blaise study does not end with Data Collection. The completed interviews are scrutinized for unusual entries, all comments are reviewed for consistency with the data entered, and all derived variables are created before processing for data delivery

is completed.

An array of systems supporting the Blaise editing will be implemented, including:

- Extract system
- Editing system
- Comment system
- Data Decision Log
- Data and interview validation
- QA reconciliation systems
- Study management systems
- Metadata system

Collected data must be made available, both for reviewing and processing. The extract system allows the data manager to access all of the data for all cases. Extraction may be performed through Manipula or Blaise API.

In the Editing process, data are reviewed and prepared for delivery. The editing system, developed in Blaise, allows the data manager to see the case data in the interview format. It displays one case at a time. The Editing System data are maintained separately from the unedited Blaise database. Decisions about delivery should be considered when initializing the Editing system which can replicate what is currently being collected in the field with the addition of derived variables, additional edits, and updates as needed.

The project flow chart will contain actions specific to the review and editing of the data. Basic steps of the editing process are:

- Combine all of the data into a master database that can be seen by the data editors,
- Maintain the Blaise interview(s), viewable in both interviewing and data editing modes,
- Display comments collected by the interview in a comment report, as well as in the interview itself,
- Data editors determine which fields may need data changed based on comments from the interviewer,
- If the data editor makes a change, the corrected skip patterns will be enforced,
- All decisions will be recorded in the data decision log,
- Other, specify fields will be coded,

- Frequencies will be reviewed, and
- Reconciliation with other forms or systems will be completed

Comment review is a major part of our editing process. Comments allow the interviewer to communicate information about particular questions that may impact the answer. The ability to view all responses associated with a comment allows the editor to follow the thought process that led the respondent and interviewer to the reviewed response. Based on the review, and following previously established coding rules, a decision is made as to whether the data entered during the interview needs to be updated or whether the remark is consistent with what was entered.

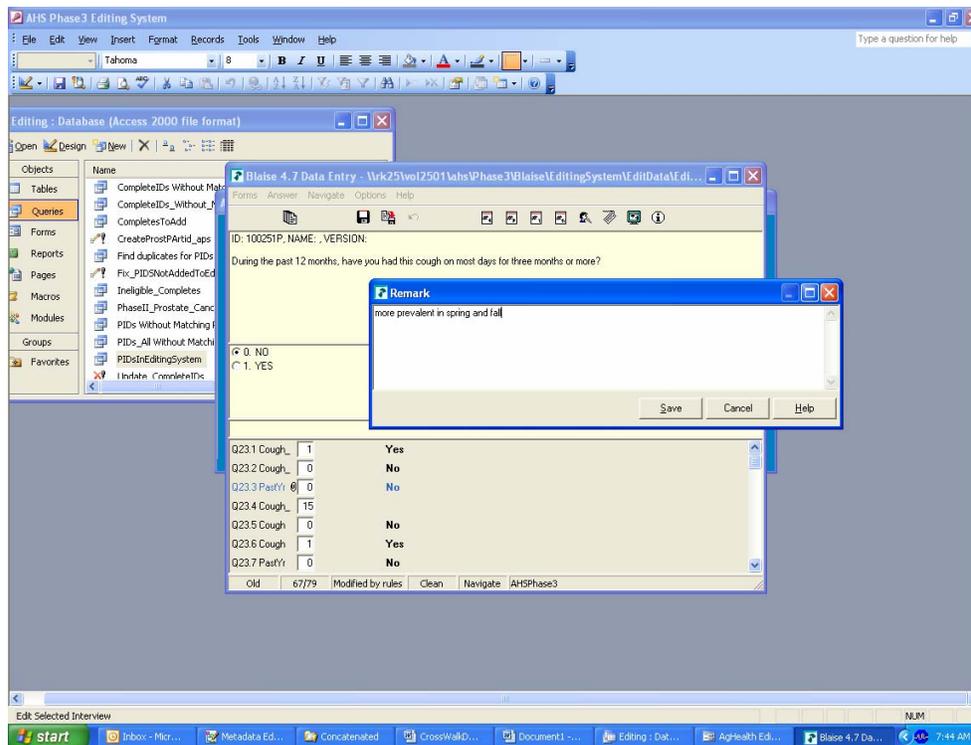


Figure 5: Comment Review

In one recent study, 36% of the interviews had at least one comment. After reviewing all comments and following the editing rules, 23% of those comments had some type of change made to the data.

Special attention has been given to the creation of a multi-faceted system which pulls the comments from the extracted data, links to the editing system, and allows for the documentation of decisions about those comments and any other data changes that are deemed necessary during the editing of the interview. The Data Decision Log is used as a tool to monitor completeness and consistency of the coding and review effort and as documentation for any decisions about the data that were made during the data processing effort. Extra fields have been added to allow for the tracking of the process, to allow for better time management.

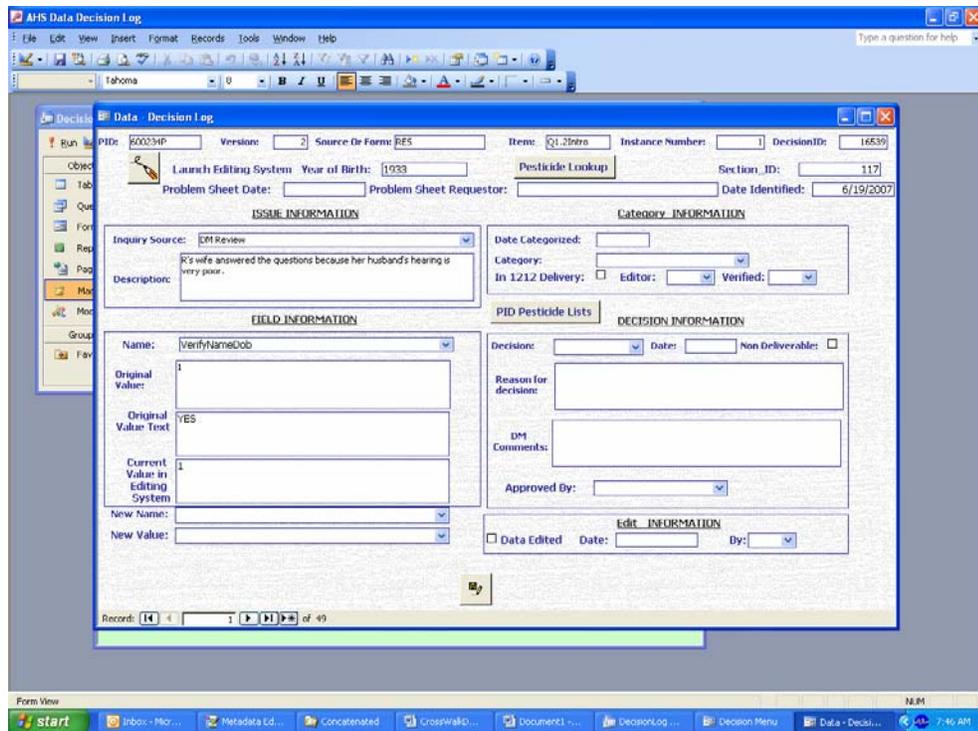


Figure 6: Inside the Data Decision Log System

All verbatim entries are dealt with. Depending on client requests, other specify fields can be re-coded into the existing code list, coded into an expanded code list, or coded as a separate variable. Specialized coding can be done manually inside the editing system or linked to it from an outside system, and completed either manually or automatically depending on the system. We have found it helpful to have separate reports for all non-numeric open-ended fields.

As a final check of the data files themselves, the data are extracted from the final edited tables and frequency reviews are performed.

There are often redundant data that are recorded in many data sources of a project. Our QA system was designed to report the discrepancies in redundant data. For example, biological sample statuses could be recorded in the Blaise Interview, the field management system, on Hardcopy forms, as well as in the Biological Repository Inventory. All of these data sources need to be reconciled. The QA system is a secured system that contains various reports based on the Editing system data, the Study Management System data, and other project source data for reconciliation. The system also has the capability of logging errors which can be suppressed on the reports. The documentation of the resolutions is concatenated with the Data Decision Log.

2.8 Stage 8 - Create Codebooks

An important part of documentation, both for internal use and for delivery, is the codebook. Each instrument requires documentation. Our codebook generation system was designed around the metadata output from the Blaise instrument. Within

the codebook generation system, one can:

- change the order of fields;
- modify question numbers, question text, and allowed values;
- add questions;
- add explanatory notes;
- add documentation of skip patterns; and
- add logic specification boxes.

Each data element is versioned, which allows for continuity of collection. All derived data elements have the associated derivation details presented in notes. Once the information is in place, one can generate a codebook without frequencies, a codebook with frequencies, or a data dictionary.

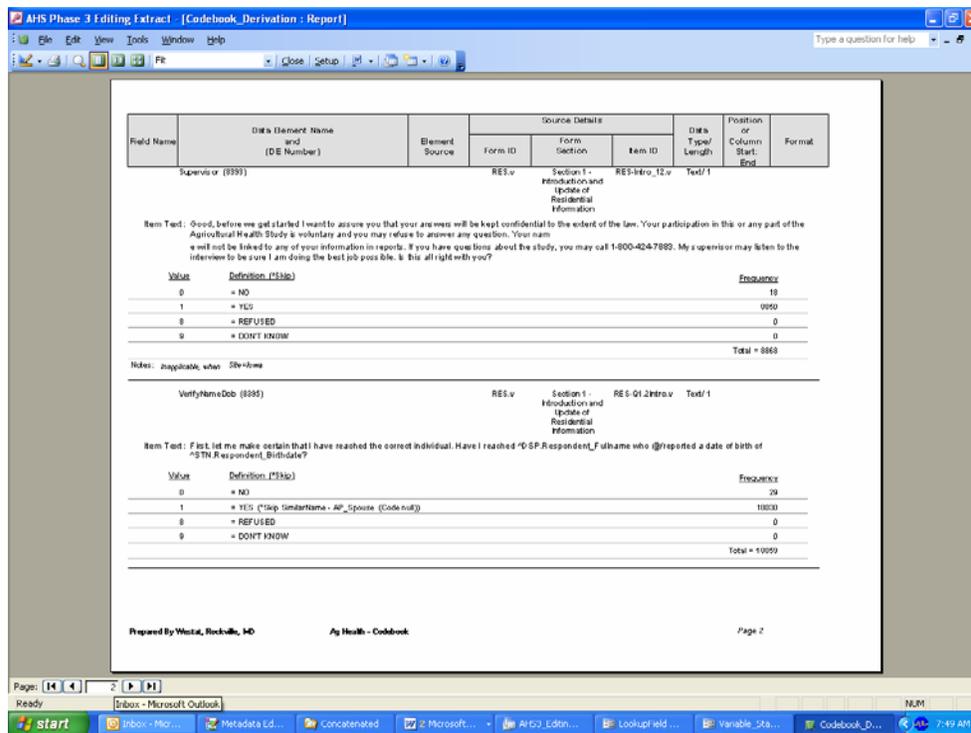


Figure 7: Codebook with Frequencies for Delivery

2.9 Stage 9 - Data Delivery

Once the data are reviewed, reconciled, and documented, delivery of both data and metadata commences. Westat has developed a Data Delivery Metadata System which supports delivery of metadata associated with datasets such as raw data, research datasets, analytic datasets, restricted use datasets, and public use files. The

system is designed to provide descriptive information about the context, quality and condition and characteristics of the data. This information helps one to understand how the data were collected and any post-collection handling that might have affected its use and interpretation. Authorized users are allowed to search, view, print, or download the metadata for which they are given access. It allows users to identify commonalities across studies, which may otherwise go unnoticed. The web-based system allows for multi-user access with strictly controlled study-specific user privileges. It is planned to remain compatible with emerging industry standards for integration and possible harmonization with external systems, and will accommodate interfaces based on common, open standards.

Projects may require only one delivery or multiple deliveries. Longitudinal studies routinely require annual, quarterly, or monthly deliveries. Complete documentation of each delivery allows for standard outputs, integrated version control, searchable linked metadata, and archival storage and tracking of metadata files.

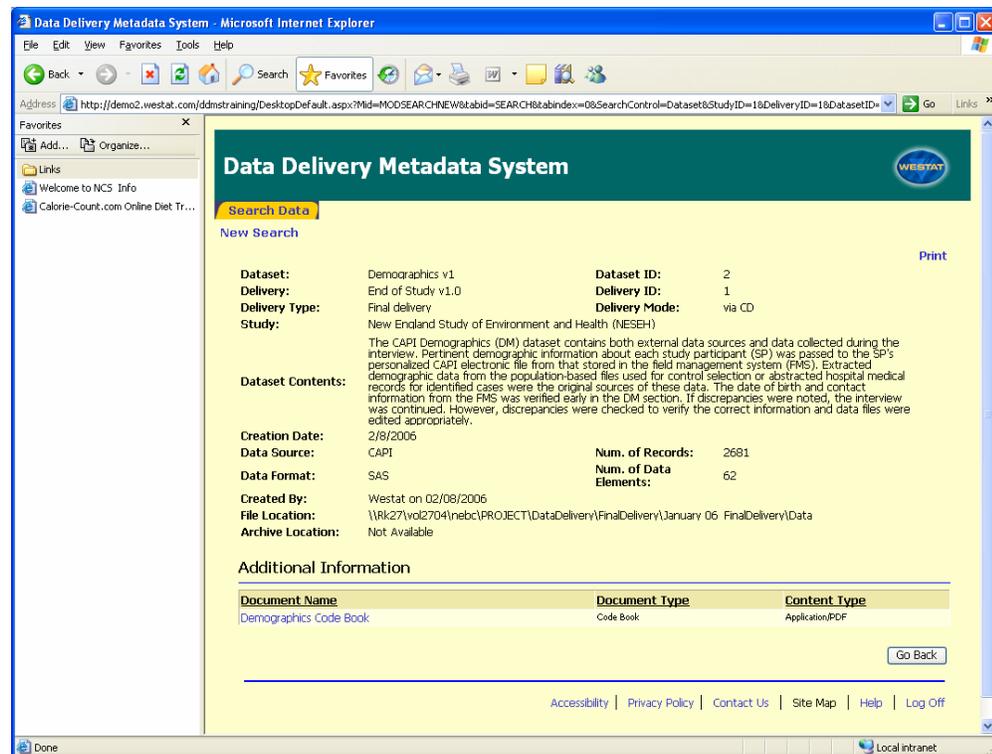


Figure 8: Data Delivery Metadata System

3. Summary

We have found that managing aspects of the processing extending across multiple lifecycle stages is critical to data quality. Feedback from all aspects of the survey need to be considered and iterative processing loops are necessary. Lifecycle tools can be implemented in multiple project environments and customized to the needs of individual surveys. The Blaise API facilitates the incorporation of lifecycle tools. Systems which emphasize ease of use and documentation capabilities allow data processing to adjust to changing needs of a study, whether internally or externally

driven. Each step of the study lifecycle is reviewed, and a system or systems chosen to match the needs of that step. This allows a specific focus and the means to identify problem areas for each process. Active participation from all team members assure that differing perspectives are represented. Quality control is a fundamental focus of each lifecycle stage and is built into the processing operations as all systems used have components relating to data quality. In this way, quality control becomes continuous from planning through delivery, ensuring accurate and efficient data collection and processing.

The ScriptWriter Tool - An Application for Interviewer Training Script Development

Youhong Liu, The University of Michigan

1. Introduction

ScriptWriter is a VB.net program that facilitates the creation of interviewer training scripts. The application allows the user to enter training annotations while progressing through a stand-a-lone version of the Blaise instrument. ScriptWriter produces .HTML output of the training annotations combined with the survey question text, category listing and responses that the user has entered. The output includes only the user's path through the survey instrument, versus all survey items, which is ideal for producing an interviewer training script. The .HTML output can easily be imported into Microsoft Word for further editing, as necessary, and saved as a Word document. The .HTML output, however, is pre-formatted and "ready to go" with minimal editing required.

2. History

Script development can often present a challenge for those involved. The difficulty of its production is primarily technical. Traditionally, script writers have hand-entered information from the Blaise screen. This consumes significant amount of time and staff resources, while increasing the likelihood of inaccurate and inconsistent entry by multiple writers. The need for a more efficient, automated approach to script development led to the creation of an application, called ScriptWriter.

3. Programming Language and Backend Database

The programming language used for the system is Visual Basic .net (VB.net). Previously, several applications were written with VB6. VB.net evolved from VB6 is a full object oriented programming language that offers several benefits, such as easy code maintenance, extensibility, and code reuse. These benefits are not found in procedural programming languages. The system uses Microsoft Access database to store and process data. Access is portable and easy to use for end users without significant training.

4. What's In the Program

In the ScriptWriter Programming directory:

- BlaiseScriptWriting.exe – Main program
- DEPSCRIPTGEN.exe – An interface called by Blaise DEP to input script notes
- Other Necessary DLLs – BCP DLLs, etc.
- Scripting.mdb – An Access database template

5. Data Model Files and Requirements

Required data model files:

- Blaise data model files – BMI, BXI and BDM. The core file for the system is the adt file that is generated during the data entry sessions. In order to obtain this file, the data model should be prepared with “Make Audit Trail Option” checked in its mode library.
- Blaise database files – BDB and other related files. It should at least be preloaded with primary keys. Some data models may require more involved preload (i.e. preload variables that drive the flow through the instrument).
- Blaise menu file – BMF. In this file, an item should be set to access DEPSCRIPTGEN.exe; e.g. making F6 as its shortcut.
- Other Support files – Lookup Blaise databases, DLLs and other files required by the data model

6. System Logic

6.1 Generate adt and mdb File

The first step to create a training script is to start entering a case in the Blaise DEP with a selected data model and preloaded database – See Figure 1. The access database is used to store training notes. The template scripting.mdb that is located in the application directory is copied to the data model directory when a new case is started in the Blaise DEP. It is renamed as [PrimaryKey].mdb.

During data entry, if the user wants to enter training notes, they can press “F6” to bring up the DEPSCRIPTGEN program interface (Figure 2). The DEP will pass the correct field name to the program. When “Save” is clicked, it will save the data to the mdb database.

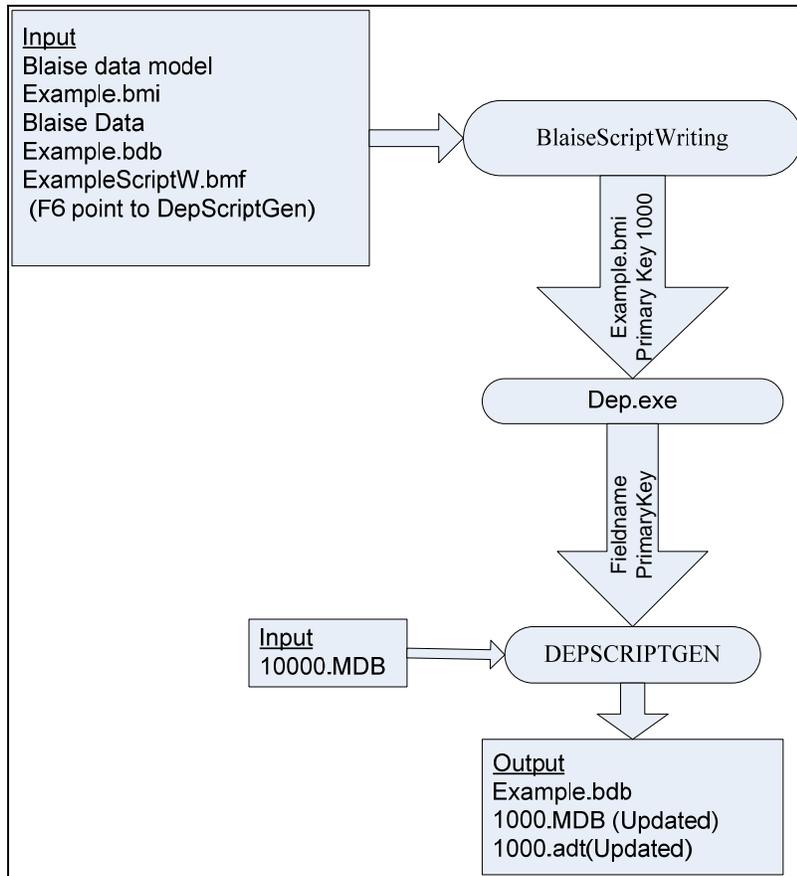


Figure 1: adt and mdb Files Flow Chart

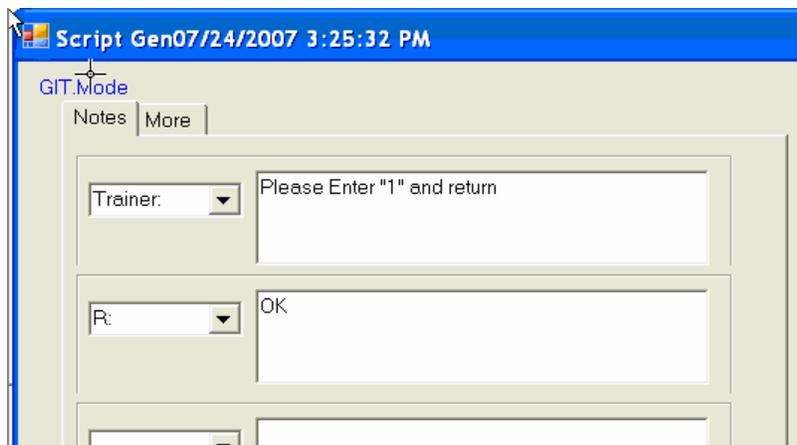


Figure 2: DEPSCRIPTGEN Interface

6.2 Editing Training Notes via ScriptWriting Program

BlaiseScriptWriting has a similar interface as that of DEPScriptGEN. See Figure 3. On the left, the field sequence from the adt is listed. The field tag, name, value and code are also displayed for reference. To edit training notes for a field, just navigate to the field and update the data on the right. This way, the user can enter or edit data outside Blaise. This is important, because if the Blaise DEP is used to launch the DEPScriptGEN every time the user wants to change notes for a field, the user will have to go back and forth to search for the field. It is not only time consuming, but also would leave a lot of unwanted entries in the adt file that would make the output processing more difficult.

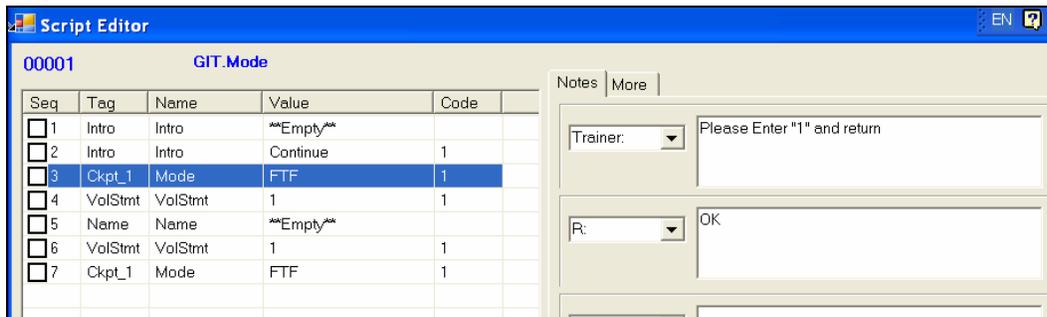


Figure 3: Editing Script Outside of DEP Session

6.3 Generating Output Html File

After the notes are all entered, the updated Access database and adt file are used to create the final script file, namely [PrimayKey].html. In Figure 5, the data items in the html file are from different sources.

Examine the contents of the file for accuracy from the html output. If necessary, the user can go back to the interface, in Figure 3, to determine if additional training notes are needed or changes required. Repeat this process until the output is satisfactory.

During html generation, a temp bdb is created to store data read from the adt file. This temp bdb is used to obtain correct question and category text. For example, if there are fills for a question dependant on responses from previous questions, the bdb with updated field must be used to obtain the correct question text. The bdb is deleted upon completion of the html file.

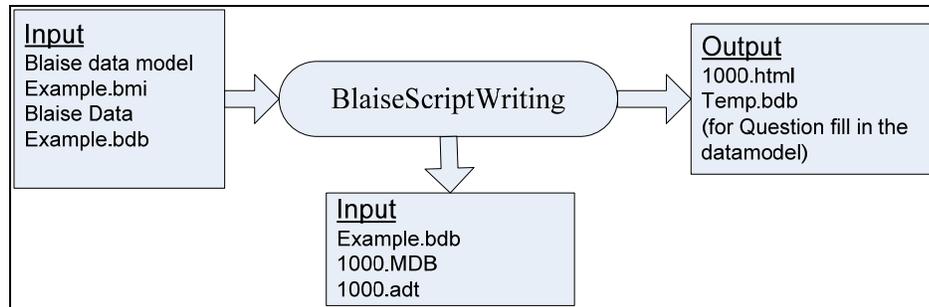


Figure 4: Generate .html Output File

(Field name from the Blaise data model)
Ckpt_1 GIT.Mode
(Field text from the Blaise database)
 Interviewer checkpoint:
Is this a face-to-face or telephone interview?
(Code categories from the Blaise database)
1. Face-to-face interview
5. Telephone interview

(Training notes from the Mdb file)
 Trainer: Please Enter "1" and return
 R: OK

(Field Value from the adt file)
Value: 1 (Face-to-face interview)

Figure 5: Sample html output

6.3 Removing Duplicate Variables from Output

Due to the nature of the ScriptWriter program, the user will need to perform a small amount of editing to account for the presence of duplicate variables either from ScriptWriter's digestion of the .ADT file, or from the user backing up to make corrections during the Blaise entry. The Seq# box of any variable can be checked to eliminate it from appearing in the script output (see Figure 6 below).

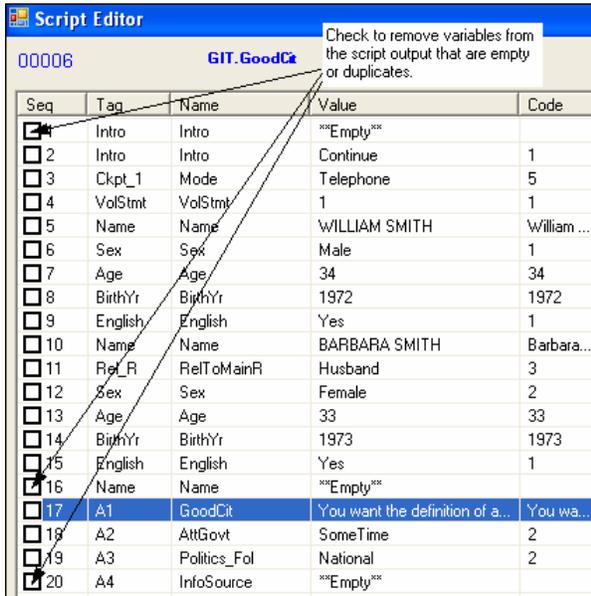


Figure 6: Removing Duplicate Variables

6.4. Script Lite

The reason to add the ScriptLite function to the system is to deal with data model changes. For instance, if fields are added to the data model and the user continues to use the adt produced by the previous data model, the output will be out of sequence (assuming the fields were added somewhere in the middle of the instrument). In this case, the whole case has to be reentered to ensure that the correct questionnaire flow sequence is in the adt. To preserve the data entered in the previous data model, the ScriptLite is used to save that data to the mdb so the user can quickly go through the case in the new data model. Note that at this point, the training notes the user spent hours inputting via the previous data model in the mdb are still available. Only training annotations for new questions needed to be added. The data is saved to tblScriptList table. A useful report in the database is designed in the mdb so users can print out the content to help them to reenter the data. See Figure 7.

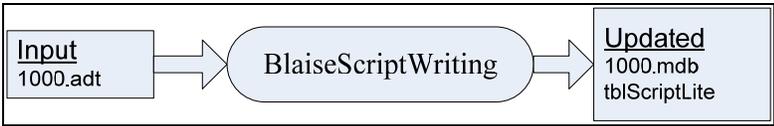


Figure 6: Script Lite Flow Chart

tblScriptLite

<i>Seq</i>	<i>FieldName</i>	<i>FieldTag</i>	<i>Code</i>	<i>CodeText</i>
2	GIT.Intro	Intro	1	Continue
3	GIT.Mode	Ckpt_1	5	Telephone
4	GIT.VolStmt	VolStmt	1	1
5	GIT.HHL.HHL[1].Name	Name	Joe Smith	JOE SMITH
6	GIT.HHL.HHL[1].Sex	Sex	1	Male
7	GIT.HHL.HHL[1].Age	Age	34	34
8	GIT.HHL.HHL[1].BirthYr	BirthYr	1972	1972
9	GIT.HHL.HHL[1].English	English	1	Yes
18	GIT.Intro	Intro		Continue
25	GIT.GoodCit	A1	Someone who votes in the elections and is aware of issues at the national and local level.	Someone who votes in the elections and is aware of issues at the national and local level.
26	GIT.AttGovt	A2	2	SomeTime
27	GIT.Politics_Fol	A3	1	International
28	GIT.InfoSource	A4	2	Magazines
29	GIT.Comp_Use	B1	5	No

Figure 7: ScriptLite Output

6.5. Importing the .HTML script Output into Microsoft Word

Finally, after users confirm that they have everything required from the ScriptWriter, the html file can be imported into Microsoft Word for further editing. The users can edit the document and adjust formatting as necessary. In some cases, the margins of the page will need to be widened.

6.6 Translate Script Notes to the Other Language

In the mdb, form LanguageTran is designed to translate the training notes to the other language, see Figure 8. In the ScriptWriter interface, select the data model language along with the “User Other Language” check box, as seen in Figure 9.

The screenshot shows the 'Script Translator' application window. The form contains the following fields and controls:

- Field Name:** GIT.Politics_Fol
- Field Order:** 19 (with a tooltip: "Corresponds with the Seq # in ScriptWriter.")
- Seq:** 1 (with a tooltip: "The sequence of the training annotation for the variable.")
- Type:** R: (with a tooltip: "Type of training annotation. Denotes 'Respondent'")
- English Notes:** I follow both internation and national politics closely.
- OLNotes:** (Empty text area)
- Record navigation:** Record: 1 of 3 (with navigation buttons)
- Record navigation:** Record: 4 of 4 (with navigation buttons)

Annotations with arrows point to the OLNotes field and the record numbers, explaining their function in the translation process.

Figure 8: LanguageTran Form

The screenshot shows the 'Output Parameters' dialog box with the following settings:

- Include Seq No.:**
- Use Other Language Field from Access (OLNOTES):**
- DataModel Language:** Default (dropdown menu)

Figure 9: Output Parameters

7. Randomization in the Blaise Datamodel

If the Blaise data model has a randomization field, and some logic and question fills depend on the value of this variable, problems may occur when using the system. This is because Blaise may generate a different value during time of script generation than it generated during time of training. In this situation, questionnaire flow and question text maybe different from computer to computer. To avoid this, it is advised to preload randomized values.

8. Conclusion

Interviewer training scripts play an integral role in Computer Assisted Interviewing (CAI) studies. They are used in interviewer trainings to provide them with practice through a Blaise questionnaire, while trainers highlight important features. Scripts are also used to assess interviewer questionnaire administration, to ensure the quality of data and to offer a means to standardize training elements. Standardization is crucial to the proper training and certification of interviewers, especially across training teams with multiple training sessions.

The ScriptWriter Tool was developed to facilitate the script development process. The goal was to create a user-friendly application with the flexibility to serve the needs of many projects and diverse Blaise applications. In most cases, using ScriptWriter should result in significant time and resource savings in the production of interviewer training scripts, as compared with alternative methods. Since the question components are pulled directly from the Blaise instrument, the resulting script is accurate and contains the components necessary to ensure the proper training and certification of interviewers.

Challenges in Converting the National Crime Victimization Survey to Blaise

Charlie Carter, Roberto Picha, Michael Mangiapane, Alexis Arrington, Daniel Moshinsky, U.S. Census Bureau

1. Introduction

The National Crime Victimization Survey (NCVS) is a major national longitudinal survey conducted by the U. S. Census Bureau for the Bureau of Justice Statistics (BJS) with a sample size of about 56,000 households annually. This survey provides personal victimization and property crime data on types and incidences of crime, monetary losses and physical injuries due to crime, along with characteristics of the victims and offenders. Although the NCVS is a household-based survey, it collects crime data for every household member who is age 12 and older. The sample is divided into six panels, with the first interview occurring in a given month and again at 6-month intervals. Each household in sample is interviewed by telephone or personal visit for a total of seven times during a three-year period.

In this paper we describe the challenges we encountered converting NCVS from a Paper and Pencil Interviewing (PAPI)/Computer-Assisted Telephone Interviewing (CATI) CASES instrument to a Computer Assisted Personal Interviewing (CAPI)/CATI Blaise instrument.

1.1 Survey Challenges

- Our first challenge was to create an approach for collecting multiple crime reports without negatively impacting instrument performance, i.e., the time delay between questions, for selecting in any order the next available household member to interview, and for navigating effortlessly between the various sections of the survey instrument.
- The second challenge involved developing an approach for retrieving details on the reported crimes and comparing them between members of the household so that the interviewer can determine if the crime is a duplicate of one that has already been reported by another household member during the current interview period as well as crimes reported six months ago during the prior enumeration.
- The third challenge with the NCVS instrument was integrating an “Editing and Coding” (E&C) instrument into the regular collection instrument and only allowing a certain set of questions for review, editing, or coding for each eligible household member, while maintaining the integrity of all of the other survey questions that were not E&C eligible.

1.2 NCVS Technical Description

The NCVS data model was developed with Blaise 4.6 software. It is a large and complex data model, containing 55 blocks and 4741 block instances. The NCVS data model produces over 60 ASCII relational output files. There are a total of 162,656 data fields with a total length of 2,289,961 bytes defined in the model. There are many different

data types and relationships in the data model. The technical description for the NCVS data model is shown in Table 1.

Table 1, Technical Description

1. Overall counts	Value
Number of uniquely defined fields	971
Number of elementary fields	924
Number of defined data fields	162656
Number of defined block fields	47
Number of defined blocks	55
Number of embedded blocks	1
Number of block instances	4741
Number of key fields	1
Number of defined answer categories	295
Total length of string fields	2131051
Total length of open fields	0
Total length of field texts	69159
Total length of value texts	39263
Number of stored signals and checks	21424
Total number of signals and checks	21424

2. Data fields	Number	Length
fields in data file		
Integer	11107	45493
Real	3	21
Enumerated	83049	87616
Set	14868	25676
Classification	0	0
Datatype	5	40
Timetype	8	64
String	53616	2131051
Open	0	-
Total in data model	162656	2289961

2. Multiple Crime Reports

The NCVS is unique among other surveys administered at the U.S. Census Bureau because of the total number of household members that are eligible to be interviewed, i.e., up to 30 members age 12 and older. A maximum of 30 crime reports can be collected for each member. If an individual respondent is not available or refuses to be interviewed, that person is either interviewed at a later date or flagged as a non-interview. Once the interviewer is successful at interviewing an eligible household member during the first enumeration, usually age 18 and older, that person is flagged as the household respondent. All of the other eligible household members are individual respondents who must also be interviewed before the survey close out date or be flagged as a non-interview.

The distinction between the household respondent and the individual respondent is important because there are multiple questions within the survey that are slated only for the household respondent. They include questions about the household composition and their characteristics, household income, housing characteristics, vandalism, and use of telephone.

Additionally, there are questions that are common to both the household respondent and the individual respondents. Those common questions include, but are not limited to, the screener questions, which include a series of probes about different categories of crimes. The screener questions are asked to report the number of incidents in the reference period and the number of times they have occurred. The household respondent and the individual respondent also share in common the crime report questions that are asked to collect details about all the crimes reported in the screener questions as well as questions about current employment and mobility [see Figure 2.1 and Figure 2.2].

For the most part, the number of crimes reported in the screener questions determine how many crime reports will need to be collected. The large number of questions and responses in both the screener and crime report blocks produce an even larger number of skip patterns because the questions are interdependent.

Our initial approach to collecting multiple crime reports for a respondent posed several challenges, as we will describe below.

2.1 Instrument Design for Multiple Crime Reports

The data collection process starts with the household respondent; the instrument collects their screening questions and crime reports, if any. Once that interview is completed the instrument takes the interviewer to the *SetupRoster* question where all members eligible for interviewing are displayed; the interviewer will then select a household member available at the time of interview. Once selected the interviewer will start interviewing until that respondent has completed their interview. The instrument will go back to *SetupRoster* allowing the interviewer to select another member or conclude the interview if all eligible members were interviewed.

In an ideal world, all of the eligible household members are at home and available to complete their interview. However, there will be situations when a respondent has to terminate their interview early, or an eligible household member may not be available for interview. The interviewer must be able to resume a partial interview with a respondent

from the previous termination point, or start an interview with another available household member.

Figure 2.1, A Question from the Screening Section.

I'm going to read some examples that will give you an idea of the kinds of crimes this study covers. As I go through them, tell me if any of these happened to you in the last 6 months, that is, since January 05, 2006. Was something belonging to YOU stolen, such as --

- ♦ Read each category

- Things that you carry, like luggage, a wallet, purse, briefcase, book -
- Clothing, jewelry, or cellphone -
- Bicycle or sports equipment -
- Things in your home - like a TV, stereo, or tools -
- Things outside your home such as a garden hose or lawn furniture -
- Things belonging to children in the household -
- Things from a vehicle, such as a package, groceries, camera, or CDs - OR
- Did anyone ATTEMPT to steal anything belonging to you?

- ♦ Ask only if necessary:

Did any incidents of this type happen to you?

Figure 2.2, A Question from the Crime Report Section.

Did this incident happen ...

- ♦ Read each category until respondent says 'yes', then enter appropriate precode.

11. In your home or lodging?

12. Near your home or lodging?

13. At, in, or near a friends's/relative's/neighbor's home?

14. At a commercial place?

15. In a parking lot or garage?

16. At school?

17. In open areas, on the street, or on public transportation?

36. Somewhere else?

2.2 Challenges of Collecting Multiple Crime Reports

In order to facilitate testing early in the development, sections of the survey had been programmed as stand-alone modules. When the instrument was first integrated, screener questions were arranged in arrayed blocks, each block corresponding to a potential eligible household member. Crime report questions were also organized into an array sized according to the maximum potential number of crimes each household member may report (30), for each of the 10 categories of eligible crimes. The resultant data model consisted of some 9000 ($30 * 10 * 30$) nested blocks, exceeded the 16,300 Blaise page limit, and failed to compile. After we reduced the size of one of the arrays associated with the maximum number of crimes allowed for each screener question, we avoided the compilation error; however, the performance remained undesirable.

The instrument load time was about 45 seconds, and there was a re-execution delay of about 2 seconds when proceeding from question to question. The poor performance was, in part, due to the large number of arrayed blocks being on route at the same time. Clearly, a conceptually different approach was called for.

2.3 Solution for Collecting Multiple Crime Reports

The survey architecture was re-designed to separate the data collection activities from data storage. Data would be collected in one non-arrayed block, and then copied into a separate arrayed storage block. This approach takes the large arrayed blocks that had slowed performance off the re-execution path, and also allows for a more flexible retrieval of partial interviews and navigation between household members.

The re-designed instrument works as follows. First, the screening questions are collected for the current respondent in the *Screening* block (see Figure 2.3). After all the screening questions are answered, the data is copied into a separate arrayed block, *NCVSScreenData* – the data storage structure for the screening questions. Then, a count of reported incidents for the respondent is calculated. If incidents have been reported, crime reports are collected in the *CrimeReport* block. After the completion of each crime report, the data is again copied to a separate arrayed storage block, called *NCVSCrimeReportData*. The interviewer can then pick another household member to interview, and the process repeats. Data collection blocks are re-used over and over again for every eligible household respondent, and for each new crime report. The data collection blocks are set to EMPTY each time their data is sent to the storage blocks, ensuring that when rules are re-executed they will be sent on route again. This iterative process continues until all crime reports have been collected and stored for each crime reported by the respondent in the Screening section – and until all available household members have been interviewed. It must also be noted that each element of the storage block arrays has the same exact field structure as the corresponding data collection block, which makes copying data from one to the other a matter of simple assignment.

Data is also copied into the storage arrays for partial interviews -- whenever the interviewer needs to schedule a follow-up interview with the respondent. Another advantage of using the storage arrays is that they allow the retrieval and continuation of a partial interview at follow-up time. Counters and flags within the instrument are employed to keep track of the household members with partial interviews, and of how many and which crime reports remain to be completed for them.

Figure 2.3, Critical Fields Used in the Data Storage Approach

Field Names	Field Definitions
NCVSScreenData	ARRAY[1..30] OF BNCVS1
Screening	BNCVS1
NCVSCrimeReportData	ARRAY[1..30] OF BNCVS2
CrimeReport	BNCVS2
SetupRoster	bSetupRoster

If the selected member cannot continue with the interview, the interviewer can setup a callback with an appointment for a later date. The interviewer can continue by selecting another member in the household from the *SetupRoster* block.

The instrument controls part of that process with the use of special person-level flags that keep track of the completion status of each household member, and, if there has been a termination, the location of the last saved crime report within the *NCVSCrimeReportData* array, and the number of crime reports that remain to be collected. In addition, each block contains identifying information (e.g. person number and incident number). These flags make possible the retrieval of any incomplete screening or crime report section. The appropriate data is retrieved from the storage block using the flags, and it is then copied into the data collection block. This functionality is critical to the implementation of un-duplication and editing and coding capabilities, as we will discuss in more details in Sections 3 and 4 of this paper.

A Brief summary of the pros and cons of this approach are listed in Figure 2.4 below.

Figure 2.4, A Summary of the Pros and Cons of the Developed Solution.

<p>Pros</p> <ul style="list-style-type: none"> • Allows the re-use of the same block as many times as needed to collect all the incident and crime reports. • Greatly improves the performance of the instrument when navigating from question to question. • Reduces the number of variables within the instrument. <p>Cons</p> <ul style="list-style-type: none"> • Prohibits returning to an incident already collected. • Complex coding for setting control flags deters changes to the rules for setting these flags.
--

3. Un-Duplication

In the course of an interview, it is likely that multiple respondents in a household will mention the same crime that happened in the last year, either in the current interview reference period of the last six months or the prior interview reference period. As part of the system requirements for the NCVS, the instrument displays certain details of the crimes to the interviewer so that they may compare the reported crimes for similarities and make decisions about whether the compared crimes are duplicates. This process is known as un-duplication and it is used to review the reported crimes to determine if they are suspected duplicates of one another. If any duplicates are found, one of the crimes

reported in the current period will be marked for deletion from the data that will be processed into the final tabulated criminal victimization data.

3.1 Un-duplication Process and Structure

The process of un-duplication of the reported crime incidents takes place after all of the incidents have been collected from the respondent. All incidents that were reported in the current interview are compared against one another with details about both crimes being displayed on the screen for comparison. If an interviewer suspects that the two displayed crime reports are duplicates, one of the crime reports is flagged by the instrument to be deleted during post-processing; then the instrument moves to the next crime report comparison. This process is repeated by comparing all of the current incidents with up to four incidents that were reported in the previous interview six months ago, if any were reported.

The main structure of the un-duplication table is a cross-reference table that only puts on-route the reported crimes from the current interview that had not been compared to one another. Each row and column represents reported crimes in the current interview. When comparing the current respondent's crime reports to all crime reports completed this interview period, the displayed rows would represent all crimes reported by all respondents, and the displayed columns would be the crimes that were reported by the respondent just interviewed. If any crime is reported as a duplicate, the crime represented in the row of the table is the one that is marked as a duplicate, and any columns past the duplicated crime are taken off-path since there is no reason to continue the comparisons. The cursor moves on to the next crime to be compared until all crimes from the current interview have been compared.

A second table is used to compare all crimes reported by the current respondent to any crimes that were reported in a previous interview. Each row represents the crimes reported by the current respondent, and each column represents a crime that was reported in the previous interview. Just like the previous table, if a crime is marked as a duplicate, any columns past the duplicated crime are taken off-path. Also, if any crime was marked as a duplicate in the previous table, it will not come up for comparison in this table. The cursor moves on to the next crime until all comparisons have been made.

3.2 Challenges of Un-duplication

The biggest challenge to un-duplication was being able to retrieve details on the reported crimes and display them in such a way that the interviewer could quickly compare the two crimes against one another to determine if one is suspected to be a duplicate of the other. To simplify the instrument for collecting the details of the crime, one block is used to collect the current crime report. After the details are collected, the report is moved into an array for storage since the interviewer does not retrieve data from storage when they're collecting the crime report. While every single detail in the crime report did not need to be displayed to the interviewer, they did need a certain information from the crime report, which included the date, location, which respondent reported it, the type of crime (theft, assault, burglary, etc.), who was there, if weapons were used, what was taken if it was a theft, what injuries there were, and any additional information that the interviewer reported in a summary question at the end of the crime report.

The immediate problem during implementation was that while we were able to retrieve the correct information from the storage array and piece it together for un-duplication, the amount of data being brought in was too large for Blaise to handle inside a single block. We were trying to use defined parameters to bring each piece of information out of the storage array into un-duplication and we exceeded Blaise's limits on the size of one block (Length of data for one block exceeds 65519 positions). The block in question was not the un-duplication block; it was the core instrument block that drives the main interview of the NCVS, including screening questions and the collection of details on the crime. There wasn't much we could do to change the core block or the other interviewing blocks, so adjustments had to be made to un-duplication.

Another challenge of un-duplication was making the screen readable to the interviewer. We needed to display some type of summary of the crime report details that the interviewer could read, it also needed to be formatted in such a way that the two crimes could be compared without forcing the interviewer to look all over the screen to compare each detail. Ideally we wanted to put the summaries on the screen next to one another vertically so that the two could be compared line by line, saving the interviewer time since they could look across the screen to make the comparison.

3.3 Solutions for Un-duplication

The initial approach of using parameters to bring in the crime information was abandoned for another method of displaying the crime details by moving the un-duplication block to a higher-level block in the instrument. A bonus of creating the un-duplication display at this level was that the un-duplication display was able to re-use some text fills that were already defined, saving the instrument the overhead of creating a number of new text fills. Tests of the maximum size of the string that would be used for displaying the crime data in un-duplication were done, and the maximum was 635 characters, which turned out to be more than enough for our purposes.

To make the un-duplication display readable to the interviewer it was calculated that the instrument could display 90 characters per line on the screen since we wanted to display the two crime reports side-by-side. A substring of 45 characters for each displayed line was created. Each line contained the data item and a buffer string of up to 45 blank characters concatenated. Blaise would automatically cut off the string at 45 characters, whether there was part of the data or this string buffer there. This was done to keep everything aligned horizontally and vertically, especially if there were blank lines because one crime description was shorter than the other. The instrument would generate one string with all of the summarized information to be brought into the un-duplication block. Inside the un-duplication block we used a direct reference to the un-duplication string that we had created in this higher-level block. Each line of the un-duplication string from each incident was brought over and concatenated together so that they were aligned. A separate block was used to create the same un-duplication display for any crimes reported in the previous enumeration that were sent on the input file, and the same process was used to put the un-duplication display strings together.

Below is an example of un-duplication screens for each un-duplication table. Un-duplication is being run for the second respondent, who reported two incidents during their interview (incidents three and four). In the example from Figure 3.1, the comparisons are being made with two incidents that were reported by the first respondent (incidents one and two). Since the interviewer answered on the first row that incident one

is a duplicate of incident three, the comparison of incident one and four is taken off-route in the table. The interviewer would proceed to the second table of un-duplication if applicable. Figure 3.2 illustrates un-duplication of crime reports three and four in the current interview being compared with three crime reports made in the NCVS interview during the previous enumeration.

Figure 3.1, Comparing Incidents Reported in the Current Enumeration

- Compare the incident data below. Do not discuss incidents with the respondent.
- Is the incident on the left a duplicate of the incident on the right?

INCIDENT REPORTED DURING CURRENT ENUMERATION	INCIDENT REPORTED DURING CURRENT ENUMERATION
Enum: 02 Ln No: 1 Inc. No: 2	Enum: 02 Ln No: 2 Inc. No: 4
MONTH: 2 WHERE: In own dwelling/att garage. Respondent and other HHLD member(s) present WEAPON: Blunt object, THREATEN: Weapon present, NOTHING TAKEN AND NO ATTEMPT TO TAKE ANYTHI SUMMARY: offender tried to break into house , was warned off, police called, offender fou nd 2 days later, replaced door	MONTH: 2 WHERE: On street. Respondent was present. WEAPON: Knife, ATTACK: Stabbed/cut w/ knife, THEFT: Cash, Wallet, Credit cards, SUMMARY: mugged in parking lot of mall, cut o n arm defending self, stitched at hospital, w allet found hours later

1. Yes
 2. No

	Undup Cur Enum								
Incident[1]			1						
Incident[2]			2						
Incident[3]									

Figure 3.2, Comparing Incidents Reported in the Previous Enumeration

- Compare the incident data below. Do not discuss incidents with the respondent.
- Is the incident on the left a duplicate of the incident on the right?

INCIDENT REPORTED DURING CURRENT ENUMERATION	INCIDENT REPORTED DURING PREVIOUS ENUMERATION
Enum: 02 Ln No: 2 Inc. No: 4	Enum: 1 Ln No: 01 Inc. No: 2
MONTH: 2 WHERE: On street. Respondent was present. WEAPON: Knife, ATTACK: Stabbed/cut w/ knife, THEFT: Cash, Wallet, Credit cards, SUMMARY: mugged in parking lot of mall, cut on arm defending self, stitched at hospital, wallet found hours later	MONTH: 09 WHERE: Commercial parking lot. WEAPON: Other personal/portable objects, SUMMARY: ART PENS STOLEN FROM PURSE IN CLASSR OOM/VALUE \$12/REPORTED TO SCHOOL OFFICIAL/NO DAMAGE DONE/OFFENDER WAS A COWORKER

1. Yes
 2. No

	Undup Prior Enum	Undup Prior Enum	Undup Prior Enum	Undup Prior Enum
Incident[2]				
Incident[3]	2	2	2	
Incident[4]	2			
[5]				

4. Editing and Coding

Editing and coding (E&C) is the process of reviewing data that was collected during interviewing for quality assurance. Criteria which determined whether the E&C instrument would be called depended on any race category set to “other” for the first time the case was interviewed [see Figure 4.1], there were one or more crime incidents reported in the household at time of interviewing [see Figure 4.2] – either telephone [CATI] and/or personal [CAPI], screener information at the person level - person called police, or person didn’t call police regarding a potential crime [see Figure 4.3].

The NCVS E&C Operation in CASES was conducted after the NCVS CATI production interviewing closeout using a separate CASES instrument and clerical editing of the PAPI forms served a comparable purpose. Based on conversations between the stakeholders and the authors, it was decided that one instrument was to be used for data collection whether CATI and/or CAPI interviewing, then the same instrument would be used for E&C of CATI and CAPI cases once regular interviewing was complete.

Figure 4.1 Race = 6 (Other)

The screenshot shows the 'National Crime Victimization Survey -- NCVS Questions ver 14.02' application window. The interface includes a menu bar with 'Forms', 'Answer', 'Navigate', 'Options', and 'Help'. Below the menu bar, there are navigation tabs for 'Main', 'HH Roster', 'FAQs', and 'Incident Review'. The main content area displays a table of household members:

LNO	NAME	REL	ORG	RACE	RACE SPECIFY
1	Ray Ray	Ref Person	No	1	
2	Teresa Ray	Wife	No	1	
3	Junior Ray	Son	No	1 6	Greek

Below the table, there is a list of race categories with checkboxes:

- 1. White
- 2. Black or African American
- 3. American Indian or Alaska Native
- 4. Asian
- 5. Native Hawaiian or Other Pacific Islander

At the bottom of the window, there is a data entry grid with columns for 'Race', 'Review Catchall1', and 'Review Catchall2'. The grid shows data for household members 2 and 3:

	Race	Review Catchall1	Review Catchall2
2	Teresa Ray		
3	Junior Ray		

The status bar at the very bottom of the window displays the following information: 00000001 | RACEROSTER_EC | 8:40:22 AM | 6-12-2007 | Talking To: | About: Ray Ray | 289/1492

Figure 4.2, Crime Incident Reports for the Household

National Crime Victimization Survey -- NCVS Questions ver 14.02

Forms Answer Navigate Options Help Show Watch Window

Main | HH Roster | FAQs | Incident Review

◆ Enter the precode to select the next incident to edit or enter 31 if finished editing all incidents

L	I	S	REF	WHAT
N	N	T	CODE	HAPPENED
O	C	A		
1	1	1	P	flower pot stolen

Review incident roster

00000001 | INCROSTER_EC | 8:42:03 AM | 6-12-2007 | Talking To: | About: Ray Ray | 291/1492

Figure 4.3, Catchall Questions.

National Crime Victimization Survey -- NCVS Questions ver 14.02

Forms Answer Navigate Options Help Show Watch Window

Main | HH Roster | FAQs | Incident Review

? [F1]

CATCHALL QUESTIONS

◆ Incident Description: **bike stolen from neighbor's yard**

◆ Select up to 3 precodes, separate with commas

<input type="checkbox"/> 11. Rape	<input type="checkbox"/> 21. Motor Vehicle Theft
<input type="checkbox"/> 12. Attempted Rape	<input type="checkbox"/> 22. Attempted Motor Vehicle Theft
<input type="checkbox"/> 13. Robbery	<input type="checkbox"/> 23. Motor Vehicle Accident
<input type="checkbox"/> 14. Attempted Robbery/Threatened Robbery	<input type="checkbox"/> 24. Vandalism (Against Household or Household Member's Property)
<input type="checkbox"/> 15. Assault	<input type="checkbox"/> 25. Prowlers/Peeping Toms
<input type="checkbox"/> 16. Attempted Assault/Threatened Assault	<input type="checkbox"/> 26. Crime Against Household, Other than above
<input type="checkbox"/> 17. Burglary	<input type="checkbox"/> 27. Not a Crime
<input type="checkbox"/> 18. Attempted Burglary	<input type="checkbox"/> 28. Crime Against Someone Else or Society
<input type="checkbox"/> 19. Larceny (Household or Personal)	<input type="checkbox"/> 29. Unable to Classify
<input type="checkbox"/> 20. Attempted Larceny (Household or Personal)	

	Race	Review Catchall1	Review Catchall2
2	Teresa Ray		
3	Junior Ray	1,2,3,4,5	

00000001 | CATCHALL2_EC | 8:41:38 AM | 6-12-2007 | Talking To: | About: Ray Ray | 289/1492

4.1 Editing & Coding Process and Structure

The NCVS E&C Blaise Instrument was the EXACT same instrument modules/blocks as the production NCVS instrument; however, it was to display only parts of the Screener section and Crime Report section, not the entire sections of each. Only the leading questions of crimes in the Screener, total number of times the crime occurred, and description of each crime were to be displayed, such as something was stolen, someone broke in, motor vehicle theft, attacked where, attacked how, sexual assault, called police crime and not called police crime. Specific data from the Crime Report section, which consisted of particular details of each crime, were to be displayed, not all of the Screener and Crime Report sections.

4.2 Challenges of Editing & Coding

A major performance issue occurred while attempting to run the instrument *and* populate the E&C block with data collected from interviewing with maximum 30 household members and maximum 30 incidents each; the instrument run time was greater than 60 seconds [one minute], which far exceeded the 20 seconds or less instrument run time anticipated for a survey.

Another performance issue occurred during the *initial* design of the E&C block whereby the block was designed as a flat file of maximum 30 household members with each household member record with maximum 30 incidents worth of data, which exceeded the block length size allowed of 32767 for any one record.

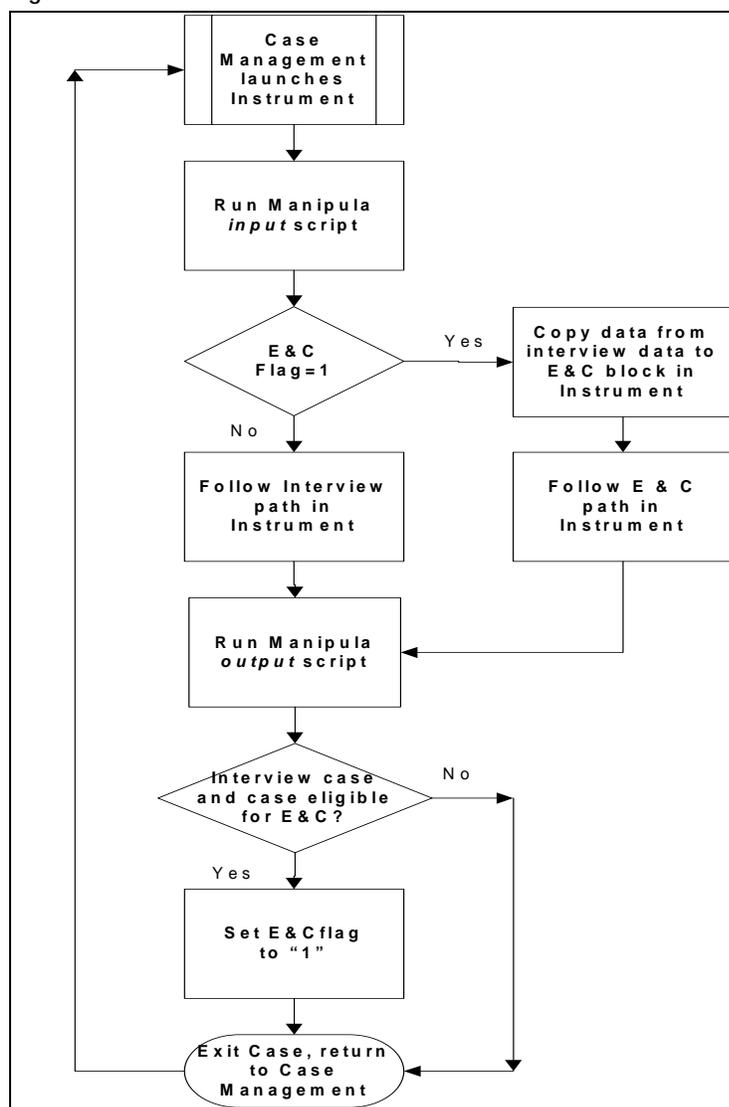
The E&C instrument was based on the premise that it would follow a different path than the production instrument. This instrument will read in a parameter that is passed in by the E&C control scripts that tells it to follow the E&C path instead of the interviewing path. In dealing with this path, although the instrument *started* with all of the production data from the original production case, some of that data was being “lost” due to the routing of the E&C instrument and off-path data for E&C questions may not appear in the Blaise databases. This challenge was how to run the E&C component of the instrument without losing the data that was collected during interviewing.

As specifications changed in the interviewing component, sometimes the same changes that were made to the interviewing instrument were not reflected in the editing and coding component which created discrepancies in screen transparencies, the flow of data and skip pattern changes just to name a few.

4.3 Solution for Editing and Coding

To resolve the performance issues – instrument run time, the E&C block would be populated with the interviewing data collected; via the Manipula *input* script *after* the E&C flag was set to “1”. This flag would be set using an Manipula *output* script *after* interviewing data collected/case complete. This was the quickest and most efficient mode to update the E&C blocks without tying up resources before calling the instrument [see Figure 4.4].

Figure 4.4 NCVS Flow Chart



To resolve the exceeding block size, a separate E&C component block was created to be a separate block within the instrument and to include only those variables that were needed from the interviewing blocks. The E&C block consisted of three tables – one table consisting of E&C household roster [see Figure 4.5]; the second table which arrayed the maximum 30 household members with race, called police crime, not called police crime, and two E&C catchall questions [see Figures 4.1 and 4.3]; and a third table that arrayed the maximum 30 crime report incidents that matched incidents to the person number and line number [see Figure 4.2]. The E&C household roster displayed pertinent data about the household member, i.e., name, age, gender, relationship, member status; the second table would be displayed *only* when at least one member of the household had a race = other *or* at least one member of the household had responded “Yes” to the called police crime *or* not called police crime questions [i.e. Catchall Questions]; the third table would be displayed when at least one crime report was recorded, then the household member data would be displayed, selecting one incident at a time along with the contents of each incident to keep the buffer size at a minimum [see Figure 4.6].

Figure 4.5 E&C Household Roster

National Crime Victimization Survey -- NCVS Questions ver 14.03

Forms Answer Navigate Options Help

Main | HH Roster | FAQs | Incident Review

NCVS Editing/Coding Roster

Control Number : 999999999999999999 Telephone Center/RO: 28

Phone Number: (800) 555-1212

Address:
100 NEW STREET ADDRESS
NEW CITY NAME, AZ 22222 1111

L		S	A	R	S	M	DATE OF	H
N		SLF/	T	G	E	E	LAST	H
0	NAME	PRXY	A	E	L	X	INTERVIEW	R
1	Ray Ray	Self DONE-Int	30	Ref Person	M	yes	7-18-2007	X
2	Wife Ray	Self DONE-Int	30	Wife	F	yes	7-18-2007	
3	Junior Ray	Self DONE-Int	13	Son	M	yes	7-18-2007	

1. Enter 1 to continue

Household

00000031 | SCREEN1 | 11:15:55 AM | 7-18-2007 | Talking To: | About: Ray Ray | 288/1492

Figure 4.6, Crime Report Summary – Beginning of E&C Crime Report

National Crime Victimization Survey -- NCVS Questions ver 14.02

Forms Answer Navigate Options Help Show Watch Window

Main | HH Roster | FAQs | Incident Review

◆ YOU ARE EDITING LINE: 1 Ray Ray /INCIDENT NUMBER : 1

◆ SUMMARY:
Summary #1 - Ray Ray, SQTheft in January, flower pot stolen from own yard

1. Enter 1 to continue

Why deleted in prod.
Review deleted incidents
Incident Summary

00000001 | SUMMARY_EC | 8:42:25 AM | 6-12-2007 | Talking To: | About: Ray Ray | 292/1492

The situation of data being “lost” due to the routing of the E&C instrument was dealt with by strategically placing keep statements at block levels within the instrument so that

no data would be lost when running the E&C component, and all other components [CAPI and CATI interviewing (data collection) blocks] were off-line/disabled. To ensure the keep statements worked properly and that data collected was actually being kept, a sample of test data were reviewed using the Watch Window debug feature, during both interviewing and E&C stages to see at what point the data was being lost, and at what point the keep statement would be most effective; block level or field level.

The problem of disabling the E&C component of the instrument without losing data collected during interviewing was accomplished via an E&C flag initialized to “blank” or “empty” within the instrument according to the transaction code in the Manipula transaction script that calls the instrument’s E&C component. The E&C component of the instrument was disabled while interviewing, but *after* the interviewing data collection was done and the case was determined to be eligible for E&C, the E&C flag was set to “1” via the Manipula output script.

With specification changes in the interviewing component, it was critical to ensure that the same changes that were made to the interviewing instrument were reflected in the editing and coding component for consistency and data review accuracy. This included, but was not limited to, screen text changes, variable name changes, and skip pattern changes. The instrument development team had to work closely and notify team members when changes were made to both components.

5. Conclusion

The NCVS is a multi-functional and complex survey that has operated continuously since 1972. The BJS uses the crime data for the dissemination of public files that are released on a semi-annual basis. When we first started converting NCVS from a PAPI/CATI CASES interview to a CAPI/CATI Blaise instrument, we encountered multiple challenges as discussed in this paper and would suggest the following recommendations for future survey efforts with a similar survey architecture.

- In order to improve performance, separate the data collection activities from the data storage when there are multiple respondents that are being asked the exact same series of questions
- Develop the E&C instrument as a second stage instrument instead of integrating it into the regular data collection instrument.

We were able to surmount all of the challenges discussed in this paper through good communication with our internal sponsor, which started by educating the sponsor about how their design decisions may adversely impact development issues such as instrument performance. That was our key to a successful deployment of the NCVS instrument into the field for data collection in July of 2006.

6. Disclaimer

The views expressed on (statistical, methodological, technical, or operational) issues are those of the author(s) and not necessarily those of the U.S. Census Bureau.

The ONS Integrated Household Survey: The next installment

Alessio Fiacco, Office for National Statistics, UK

1. Introduction

The UK's Office for National Statistics (ONS) is engaged in an extensive modernisation programme. This includes a statistical modernisation project that will bring about the integration of some of its continuous household surveys.

The aim of the project is to deliver better value for money while increasing the value of statistical outputs and providing a flexible means of meeting future survey needs (Bumpstead 2004 and Burrell 2006).

The process of 'integration' includes the whole of the survey process, including data collection, processing and delivery of outputs. This paper will consider some of the Blaise issues faced in the implementation of the ONS Integrated Household Survey (IHS), focusing on the survey instrument strategy adopted.

2. Current plans

Original plans for the IHS envisaged a January 2008 'go live' date for all component surveys. More recently, it was decided to have a phased implementation of the IHS in 2008, in order to better manage the risks associated with the 'big bang' approach.

From January 2008, the General Household Survey (GHS), the Expenditure and Food Survey (EFS) and the ONS Omnibus Survey will be integrated into one household survey.

The integration of the Labour Force Survey (LFS) into the IHS and the introduction of the unclustered sample design (for all non-LFS survey streams)¹ will occur later in 2008, to coincide with the delivery of the new survey case management system. This is the system required to handle the flow of survey information and manage field operations on a fully unclustered sample.

Earlier this year, ONS made a successful bid to carry out the new English Housing Survey for the Communities and Local Government department. This housing survey will come within the IHS framework from April 2008, in conjunction with the implementation of the LFS into the IHS.

¹ The LFS is already based on an unclustered sample design.

Table 1. IHS component surveys and related acronyms

IHS Survey	Formerly known as	IHS topic (acronym)
Labour Force Survey and associated boosts in England, Wales and Scotland	LFS / LLFS	Work (WRK)
Expenditure and Food Survey	EFS	Living Costs & Food (LCF)
General Household Survey	GSL	Lifestyle (GLF)
Omnibus Survey	OMN	General Opinions (OPN)
English Housing Survey	n/a ¹	Housing (HSG)

¹ This is a new survey. It is a merger of two existing surveys, the Survey of English Housing and the English House Condition Survey.

The IHS questionnaire will include all the information required by the component surveys, which will therefore be discontinued in their current form. The questionnaire consists of a common ('Core') module, providing information on census-type socio-demographic variables. The Core will be administered to the whole sample, while different topic modules will be administered only to parts of the sample.

In July 2007, ONS also won the contract for the new Longitudinal Disability Survey of Great Britain. This longitudinal study, which is commissioned by the Office for Disability Issues, is likely to be brought into scope and launched as an IHS survey, when fieldwork starts in April 2009.

Table 1. Brief history of the IHS

Date	Event	Blaise strategy
2004-05	Consultation exercise (talks, presentations and seminars) Meetings with key stakeholders; publication of articles and presentation of papers to national and international conferences	n/a
December 2004	Prototype test	Single survey instrument
Spring/Summer 2005	Field trials carried out	Single survey instrument
February/March 2006	First pilot survey	Single survey instrument
April 2006	Integration of Field Force and Field managers	n/a
January 2007	Successful bid for English Housing Survey	n/a
February/March 2007	IHS systems test/'Dry-run' (in-house)	Four separate survey instruments ¹
June/July 2007	Second pilot survey of over 3,300 households	Five ² separate survey instruments
September/October 2007	Longitudinal modules second wave (Work) and 'dry-run' (Lifestyles)	Separate survey instruments
April 2009	Longitudinal Disability Survey	Separate survey instrument

¹ The Blaise datamodel were built from both common and survey-specific blocks

² Additional datamodels were used (e.g. LCF Editing and Coding version: LCF expenditure Diary etc.)

3. Survey instrument strategy adopted

Previous IHS field trials and pilot work were carried out using a single Blaise instrument that included a Core set of questions and all questions for all topics. The routing to the survey-specific sections was based on sample record information.

In more recent questionnaire piloting, a multiple instrument approach was adopted, which involved using five Blaise datamodels (one for each interview type). The same common

code (e.g. Core questions) was included in each questionnaire to avoid duplication of effort and obtain consistency of data.

The technical decisions made in relation to the IHS were influenced by the constraints imposed by legacy systems, as well as other considerations such as the need to minimise the impact of change.

There were several factors that were taken into account, as the IHS is an end-to-end process and the choice of the survey instrument strategy has an impact on other aspects such as field management, data processing, business organisation to name a few. These factors are explored in detail below.

3.1 Field management

Data that informs field management operations (Case Management System reports) currently rely on the use of separate Blaise instruments in order to better monitor response and the assignment status on the IHS component surveys.

For historical reasons, there has been a strong culture of survey independence in the ONS, which resulted in its case management system being geared to the needs of its many diverse projects (Hofman and Gray 1998).

Even with the launch of the IHS there will still be a need to provide data on topic, as well as the IHS as a whole, to meet different survey requirements.

Response rates on the Core will be monitored by the use of additional Core-only outcome codes.

3.2 Data processing

The need to reproduce current outputs was identified as one of the main concerns in terms of changing the survey instrument approach, compared to early IHS pilot work. At present, various processing systems are used by different IHS component surveys. In addition, the component streams have diverse processing and data turnaround cycles.

For example, the WRK stream is a weekly survey that produces rolling monthly outputs and has a very short period for interviewing reissues (so-called ‘hangover week’). Non-WRK modules, being monthly surveys, have generally longer field and reissues periods and - as in the case of LCF – the whole survey year could potentially be used to carry out reissues.

The WRK module is also unique compared to the other IHS streams, in that it uses SIR-based processing in the production of its derived variables. Non-WRK streams perform edits, derived variables computation in SPSS or Manipula. The use of a single Blaise datamodel for the IHS would require all survey streams to be processed in the SIR database-package at some stage.

For WRK, to continue to provide current survey outputs, all redundant variables (i.e. non-WRK-related) would need to be dropped before going into SIR. Non-WRK surveys would require more effort to reproduce current outputs, as their data are not currently processed in SIR. On both accounts, the multi-instrument approach in Blaise was

deemed preferable.

It was thought that having a single IHS Blaise database, created from the single instrument, could potentially introduce a bottleneck when processing the data. It would also cause the processing cycles for all survey streams to become interdependent.

3.3 Business organisation

In preparation for the recent IHS pilot work a Core IHS team has been established. This team is responsible for the delivery of Core outputs, such as tables and microdata files. The Core team will also be responsible for delivering and coordinating changes in the Core section of the Blaise questionnaire.

Existing survey teams will continue to exist after the launch of the IHS and will be responsible for implementing changes to topic-specific sections in Blaise and output production.

To ensure that inputs/outputs are in line with standardisation and best practice, it is envisaged that the Core team and the Blaise Development, Standards and Support team will work closely, being organisationally linked to each other.

3.4 Legacy/Structural factors

Certain blocks - used for storing sampling or geographical data – may require tweaking on different survey streams. Due to the panel element of the WRK, more fields form part of its primary key compared to the other IHS modules. For instance, as part of the WRK serial number, there is a field that identifies the wave at which the address first came into the sample. This field is not present on the other IHS modules.

This prevents identical blocks from being used where the ‘key’ for searching an external file is the serial number. Thus, two versions of the same blocks may be required to accommodate searches on different primary keys.

Although, a technical solution could have been found, the use of multiple datamodels provided a suitable workaround.

Table 3. Primary key fields on IHS survey streams

Survey stream(s)	External data file search
WRK	IF PAFData.SEARCH (QID.Quota, QID.Week, QID.W1Yr, QID.Qrtr, QID.Address) ¹ THEN PAFData.READ SURVEYYEAR:=PAFData.QGEO.SURVEYYEAR (...)
GLF, OPN, LCF, HSG	IF PAFData.SEARCH (QID.Area, QID.Address, QID.Hhold) THEN PAFData.READ SURVEYYEAR:=PAFData.QGEO.SURVEYYEAR (...)

¹ Three additional fields form part of the WRK primary key. These fields are not used when matching sample record information

Another legacy issue is related to the reference period on different interview types. Historically, the survey design for the UK LFS has been based on 'fieldwork quarters'.

This means, for example, that the first week of any quarter is the first week of fieldwork for the quarter and the data collected actually refer to the last week of the previous quarter (the question text reads: “in the seven days ending Sunday {*the week before*}...”).

For the Core section of the IHS, the reference period on non-WRK surveys is based on the last seven days from the date when the interview started, whereas on WRK the date is set in advance of the case being sent out to interviewers. On the topic-specific section of other IHS component surveys the reference periods differ by question.

3.5 Time series

A range of new outputs from the whole sample Core module will be produced. However, there is a need to continue to provide survey outputs that are currently produced from the separate surveys.

To preserve the integrity of key time series, it was necessary to take a different approach on some IHS survey streams, as in the case of the Education section of the questionnaire.

The opinion module, for instance, does not require information on Educational Qualifications to the same level of detail as the other component surveys (i.e. WRK and GLF). Thus, a shorter, Core education block is used on OPN, whereas a longer, survey-specific, version is used on GLF.

4. Issues and considerations

The survey instrument decision had to take into account emergency situations, such as discovering a routing error (in any of the interview type) that requires the interviewers to download an amended version of the Blaise questionnaire.

In the design of the Blaise instrument, importance was placed on the need to minimise the impact of rescattering on all survey streams in view of potential overload on the current data transfer system.

4.1 Handling of errors

The IHS component surveys are generally fairly stable and changes are mainly introduced to incorporate new requirements at the beginning of the survey year or reflect changes in legislation, such as the introduction of new State benefits in the new financial year. The Core of the IHS is also fairly stable and the chances that the Core questions are incorrect and need rescattering are deemed very low.

The IHS includes the ONS Omnibus, a multipurpose survey that has a fast turnaround and a monthly changing set of questions/topics. Prior to data collection, the Blaise instrument is tested, however, it is a fair assumption that it is not as stable compared to the other continuous survey part of the IHS.

Using a single datamodel means that, if a routing error is found in one of the survey stream, a new (amended) instrument would have to be rescattered. Thus, the impact of a routing error, say on OPN, would affect all other survey streams.

The impact will have implications not only in terms of rescattering but also from a data

processing point of view, as extra Blaise databases would have to be merged into a single datastore, hence requiring additional computing effort.

One of the benefits of the multi-datamodels approach lies in the fact that, if errors are found in the module-specific section, only the stream affected would have to be downloaded again by interviewers and the impact on other IHS modules is thus minimised.

4.2 Context

Interviewers at present transmit their work to headquarters via internal laptop modems through a standard telephone line. The WRK is a quarterly survey whereas the other IHS survey streams are monthly surveys. Non-WRK surveys require a new questionnaire to be downloaded by interviewers every month.

Having five separate datamodels that needed to be scattered at the same time, caused some bottlenecks due to the long download times at the recent pilot (June-July 2007). The size of the questionnaires compounded the transmission problems.

Together with the large questionnaires, ONS transmits to interviewers a large WinHelp-based Help file and a large number of look-ups/coding frames. Files are sent to interviewers compressed through WINZIP.

The table below gives an indication of file sizes in kilobytes (KB) at the June/July 2007 pilot:

<ul style="list-style-type: none">• IHS Lookups: 1768 KB• IHS Help files: 1152 KB• GLF*: 1614 KB• LCF*: 1417 KB• WRK*: 1323 KB• HSG*: 631 KB• OPN*: 519 KB
--

* This included Blaise metadata (*.bmi, *.bxi and *.bdm), survey-specific lookups and sample record.

Having multiple datamodels (built from a combination of common and survey-specific blocks) may put a heavier load on the transfers system if scattered at the same time. However, it also enables a staggered transmission of the questionnaires to interviewers, as a means for reducing download times for interviewers.

ONS is working on a project, which aims to deploy the use of Broadband or possibly '3G' technology² for field interviewers. Broadband has already been implemented for Field and Regional Managers to allow faster connection to Head Office, which will be required for work allocation processes.

The Northern Ireland Research and Statistics Agency (NISRA) shares ONS's concerns regarding file sizes. NISRA is currently responsible for carrying out the Northern Irish component of three of the five IHS component surveys. Using separate Blaise

² The third generation (3G) mobile phone technology allows high-speed data transmission services in a mobile environment.

instruments was also seen as preferable for our colleagues at NISRA.

4.3 Survey-specific requirements

Some IHS component surveys require additional Blaise instruments. For instance, the LCF has a version of the datamodel used for editing and coding purposes, as well as an expenditure Diary datamodel. The GLF also has a separate instrument for its keeping-in-touch exercise (Setchfield 2007).

A single IHS mega-instrument could have been implemented to cater for all these purposes. However, such an approach would be in conflict with one of the requirements in terms of Blaise programming at ONS: to keep things simple wherever possible.

The issue of the size of the questionnaires (mentioned above) also caused a rethink on the data rotation method to be used on the WRK stream. The process of data rotation is another survey-specific requirement, which may have an impact on the data transfer system, as well as add to the complexity of the Blaise code.

For WRK, it was originally envisaged to use a Blaise database as an external file for moving the data from one wave of interviewing to the next one. This solution suited the multiple-instrument approach for the IHS.

However, it also aggravated the issue of file sizes for the WRK questionnaire. In addition, rotation issues - caused by the appropriate rules not being re-executed - were experienced, which resulted in the data being 'fed-forward' incorrectly.

Table 4. Data rotation issues when using an external Blaise database

D:\Alessio\LFS std\Source\LFS0604n - Database Browser					
	DifJob	AddJob	LookM[1]	LookM[2]	LookM[3]
	No	NewJob	FillTime	WantLong	

{The field *AddJob* should only be on route if *DifJob*=Yes, but due to the rotation method used this did not happen}

RULES {Block level}
DifJob
IF (DifJob = Yes) THEN
 Addjob
 IF Addjob = NewJob THEN
 LookM
 ENDIF
ENDIF

RULES {Table level}
{The following code is within the household array and is placed after the Block QLookWrk has been called.
QHolding is the block within the Datamodel that receives data from the external file}
IF QLookWrk[i].DifJob = EMPTY THEN
 QLookWrk[i].DifJob:=QHolding.QHHHold[i].xDifJob
ENDIF
IF QLookWrk[i].AddJob = EMPTY THEN
 QLookWrk[i].AddJob:=QHolding.QHHHold[i].xAddJob
ENDIF
{etc.}

To resolve the difficulty with the size of the questionnaire and the incorrect data rotation, a different strategy was proposed: pre-loading the fields in Blaise using a Manipula script that will be run in the WRK production line.

This method will also allow better testing of the questionnaire, reduce the amount of complex Blaise code and minimise the impact on the data transfer system.

Table 4. Pros and Cons: Single Vs Multiple Blaise instruments

Issue	Strategy	
	Single IHS instrument (February/March 2006)	Multiple IHS instruments (June/July 2007)
Impact of errors discovered after scattering:		
- Error in the common Core section ²	Rescatter once (every stream affected)	Rescatter five times ¹ (every stream affected)
- Error in the survey-specific question blocks ³	Rescatter once (every stream affected)	Rescatter once (one stream affected)
Scattering/transfer system:	Staggered file transfer is not feasible	Staggered file transfer is feasible
Download times:	2.5 hours	30 minutes per instrument
Data processing of survey stream data:	Processing cycles become interdependent	Processing cycles need not change

¹ Five datamodels were used in the second IHS Pilot (June-July 2007)

² The Core section consists of questions administered to the whole sample

³ This section is administered to parts of the sample

5. Options considered

The use of Prepare Directives to reduce the size of Blaise instruments and streamline a large number of Blaise instruments into one has been used in other organisations (e.g. Schou 2004). This option was considered for the IHS, however it was not deemed to meet the requirements of the survey.

One of the reasons for not adopting the Prepare Directive was that switching to a different directive would not create a separate metadata file (*.BMI) and data file (*.BDB): these would simply continue to be named according to the *.BLA file name.

This is a major drawback in terms of some of the survey-specific requirements mentioned above. For instance, the LCF instrument used by Coders and Editors has a different data shape (in terms of attributes etc.) from that used by interviewers.

Another reason why the use of Prepare Directives was discounted was that there may need to delete a Blaise database if the questionnaire was previously compiled using a different conditional define. This seemed to add a potential element of confusion.

6. Conclusions

Seen together with the two preceding IBUC papers on IHS (Bumpstead 2004 and Burrell 2006), this third paper in the series shows that the IHS is still evolving and how ONS has begun to adapt some of its earlier theoretical approach in the light of constraints posed by business processes.

There are various considerations when deciding which survey instrument design to follow. In the case of the IHS, end-to-end survey processing influenced some of the technical decisions on which Blaise strategy to adopt. Reducing the impact of change, in view to safeguard the ability to reproduce current outputs, for instance, had attached a high priority at ONS.

The phased-implementation of the IHS in 2008 should bring benefits in terms of the lower risk that this approach brings. The IHS will consist of five different questionnaires that are designed on a modular basis, being built around a common Core questionnaire. New survey questionnaires can be easily added on to the Core questionnaire and brought into scope of the IHS.

The introduction of the HSG module, which was not part of the first pilot (in 2006), is an example of the flexibility of this system and shows the feasibility of adding modular surveys to the IHS. As the IHS becomes established and new surveys are brought into scope, the IHS importance as a key household survey for UK official statistics is also being confirmed.

The IHS data collection design does not result in duplication of code maintenance as far as Core questions are concerned. In the longer term, the IHS should also bring about greater harmonisation on some topic-specific questions across different interview types (e.g. income questions on LCF and GLF).

Although, five different Blaise questionnaires were used in the recent pilot work, they all shared the same Mode Library to ensure consistency of screen layout. In addition, the use of the Question-by-Question Help files on IHS ensures that same working practices extend to the interviewing process on all the IHS component surveys (Setchfield 2006).

The chosen design for the IHS means that version control to ensure consistency of changes across the IHS-based surveys is of paramount importance. Version control needs to be managed using configuration control methods, such as using the latest version number as part of file names. The use of the network path for Core/common include files in Blaise is a possible way, although this was not tested in the June-July 2007 pilot. Having a dedicated Core Blaise team should ensure that version control and changes are managed in a coherent way.

Technical solutions to some of the Blaise issues linked to the use of a single Blaise instrument exist. This paper has merely scratched the surface in terms of the complexities of dealing with such a mega-instrument.

There is no doubt that some fine-tuning is required before IHS goes live in 2008. Work to address some of the issues arising from the recent pilot work is ongoing. By the time the next IBUC meets the IHS will be live and running. At that point we should have a clear overview of the whole development and, perhaps, one more but final paper on the subject.

7. References

Bumpstead R. (2004) Integration of UK government household surveys: Development of a modular Blaise instrument for a proposed Continuous Population Survey. International Blaise User Conference 2004

Burrell, T. (2006) Developing an Integrated Household Survey (IHS) Blaise Questionnaire from Four Major Social Surveys. International Blaise User Conference 2006

Hofman, L., and Gray, J. (1998) CAPI Survey Management Systems: Case Management on Laptops. In: Couper et al. (eds.), Computer Assisted Survey Information Collection.

Schou, R. (2004) Moving 43 Generated Unique Instruments to One National Instrument. International Blaise User Conference 2004

Setchfield, C. (2006) Training and Learning Opportunities using Blaise. International Blaise User Conference 2006

Setchfield, C. (2007) Coping with people who just won't stay put: The Use of Blaise in Longitudinal Panel Surveys. To be presented at the 11th International Blaise Users Conference, Annapolis, September 2007

Questionnaire Specification Database for Blaise Surveys

Lilia Filippenko, Valentina Grouverman, Joseph Nofziger, RTI International

1 Introduction

Developing large scale, complex Computer Assisted Interview (CAI) instruments is a team effort that presents many challenges. When a group of specification writers, translators, programmers, and testers are working in parallel, specifications can quickly get out of sync with program code. The Questionnaire Specifications Database (QSD) facilitates all facets of the CAI development life cycle and streamlines the process of creating Blaise instruments, especially those with a second language and/or Audio Computer Assisted Self Interview (ACASI).

More than a common code generator, QSD allows spec writers and translators to make iterative changes at any point in development, while maintaining the integrity of the instrument. Changes to wording, question fills, and response types are inserted directly into the Blaise instruments. QSD produces specification documents that are complete with question text, response options, fills, and routing comments. Since these specifications and the current Blaise code are both generated from QSD, they are always in sync. Extensive support for ACASI includes script generation, audio review, and synchronization of code with audio files. Issue tracking and change logging are built-in features.

2. Life Cycle and Changing Requirements

QSD supports the full life cycle of large scale instrument development, beginning with the specification of requirements. The questionnaires in such efforts may have multiple sections, a high degree of complexity, multiple languages, or a combination of these factors. Thus, instrument development requires a team effort. Each member or subgroup of the team has a stake in the outcome, and therefore influences the requirements.

Changing requirements are a reality of software development, and instrumentation is no exception. In a typical scenario, the primary source of requirements is the client, or possibly more than one client. Managers may be forced to demand cuts based on budget constraints or schedule. Survey specialists create programmer-ready specs, consulting methodologists for refinements and ensuring usability. Review boards, internal and sometimes external, may need to approve the specifications. This scale of development often demands multiple programmers, who discover inconsistencies or suggest changes to simplify coding. Thus begins an iterative cycle of spec writing, coding, review, testing, and revision involving all these parties and more, from quality assurance to translators.

Figure 1. Questionnaire requirements come from many sources.

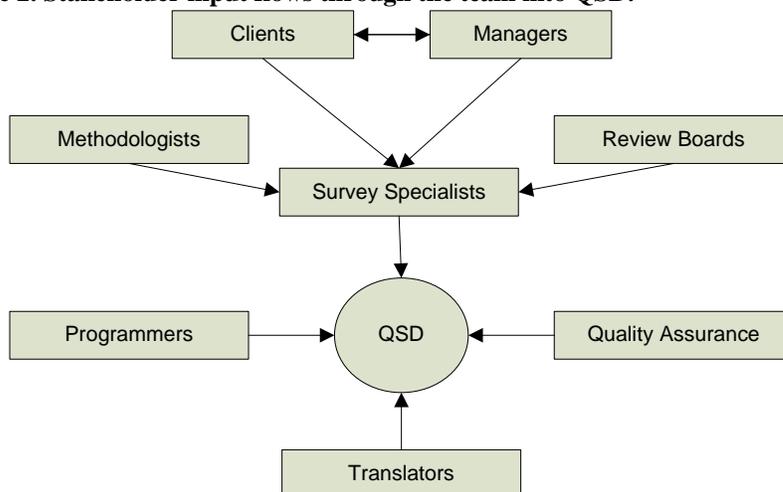
Stakeholder	Focus
Clients	Elicit complete, correct, consistent data
Managers	Balancing client needs with budget and schedule
Survey Specialists	Interviewer usability and training
Methodologists	Bias in wording; Visual representation; User interface
Review Boards	Respondent protections; Compliance; Sensitivity
Quality Assurance	Function versus specs
Translators	Language constructs
Programmers	Correct function; Adherence to specifications

QSD helps to keep the task manageable. The key to accomplishing this is the structure it imposes onto the program code. This structure, described below, facilitates quick incorporation of the most common changes we encounter: wording changes. These can be entered by non-programmers and incorporated immediately into a working instrument without fear of side effects or coding errors. This speeds the development cycle by quickly getting the new version to the testers or back out to production.

For multi-language surveys, not only question text must be translated, but also every type and fill. Translators see all question elements together and enter their translations, which are included in the generated code without any programming. A comprehensive view of untranslated questionnaire elements helps ensure complete coverage.

The stakeholders, with their various perspectives, benefit from customized views of the specifications. Since the question definitions are edited only through QSD, its contents are guaranteed to match the instrument, making production of accurate spec documents at any point in the process a trivial task.

Figure 2. Stakeholder input flows through the team into QSD.



All entered changes are logged. Problems found in testing are recorded directly in QSD.

In the next sections we explore in greater detail how users interact with the system.

3 Using QSD

QSD consists of tiers for the user interface and business rules, and a back end relational database. Microsoft Access was chosen based on its simplicity, its familiarity among the programmers, and to ease the transition from working with specifications exclusively in word processing documents to using a central system.

Each survey team uses a dedicated instance of QSD. Specific information such as instrument names, languages, and use of ACASI, is saved in a survey configuration table. User access is managed internally.

Users log in to the system using their network username and a QSD-specific password. After being verified as authorized on the survey, the user is presented with several options.

Figure 3. From QSD's initial screen the user selects an instrument, a module, and a function to perform.

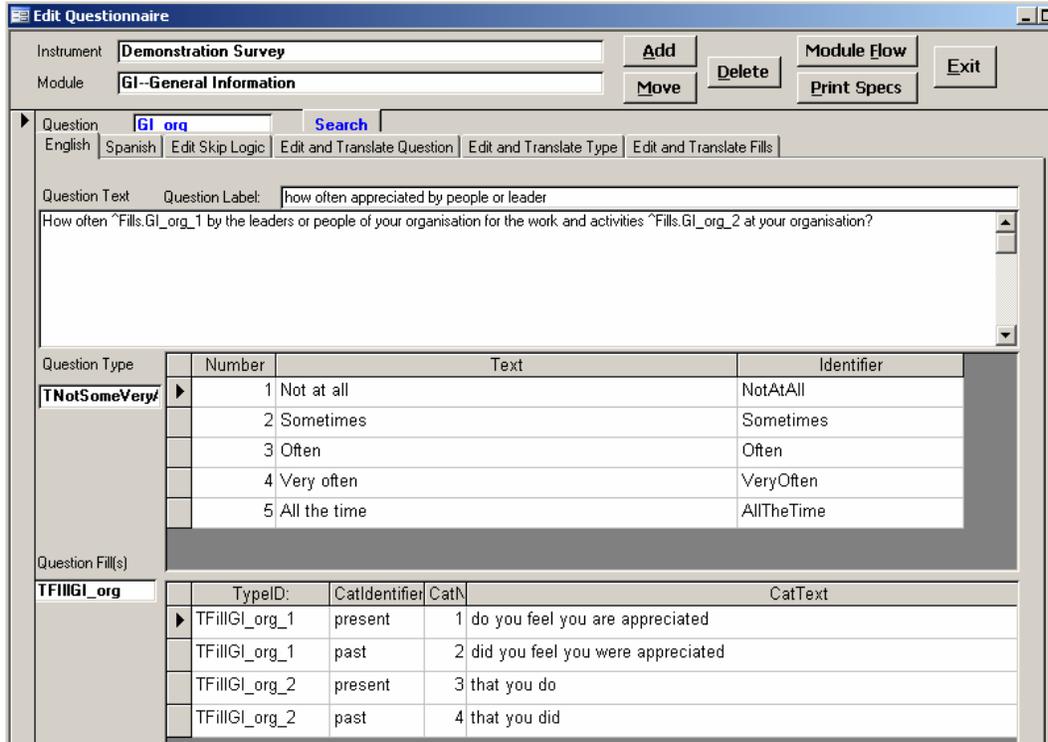
The screenshot shows a window titled "Select Module" with a blue header bar. Below the header, there is a blue instruction: "Please select Instrument and Module names from the dropdown list." Two dropdown menus are present: "Instrument Name" and "Module Name". Below these are three buttons: "Questionnaire", "RCD Developers", and "Exit". A large container holds several other buttons: "Preview Specs for Module", "Preview Specs for Instrument", "Save Specs for Module in a File", "Save Specs for Instrument in a File", "Verifying Changes to Blaise Code", "Changes to Comments by Module", "Changes to Specs by Questions", "Changes to Comments by Date", "Edit or Add New Module", and "Preview Types for All Instruments". At the bottom center is a button labeled "Manage Translation".

On the "Select Module" form, various actions are available depending on the user's access level. From here the specs can be viewed as a Microsoft Access report, or exported to a Word document. Entered spec changes can be reviewed directly in the changes table or printed as reports in a more readable format. The "Manage Translation" option leads to additional functions for translators.

When requirements for a module are changed or a module is ready for translation, the user selects "Questionnaire" from the "Select Module" form. The resulting "Edit

Questionnaire” form is where all changes to the specs for the selected module are entered.

Figure 4. On a single screen, users can change wording and translate, add, move or delete questions.



3.1 User Levels

Since many people with different needs have access to QSD, we want to be sure that only appropriate options are available to each user. There are three levels of users in QSD:

- Standard users;
- Questionnaire spec writers;
- Questionnaire programmers.

Translators and Testers are given access as “Standard user”. Survey specialists have “Spec writer” access.

Translators can see all pages on the form “Edit Questionnaire” but they can make changes only on the pages “Edit and Translate Question”, “Edit and Translate Type”, and “Edit and Translate Fills”.

In addition to standard functions, survey specialists can modify the text description of the skip logic for a question and add or change question attributes. If there is a need to add a new response option to a type or a new fill to a question, a request is added to a field named “Comments” on the “Edit Skip Logic” page. This field holds information about all changes made to the question during development of the instrument. Programmers will use this information to complete the implementation.

Our goal is to partition responsibility in instrument development by allowing the spec writers and translators to make wording changes directly, thus freeing programmers to concentrate on programming the instrument flow. The button “RCD Developers,” on the “Select Module” screen referring to RTI International’s Research Computing Division, is available only to programmers. A programmer can use QSD as follows:

- Load modules (sections), question types with responses, and question fills into QSD from a Microsoft Word document;
- Add new response options (question types) and new question fills;
- Export changes made by specs writers and translators into Blaise code;
- Create scripts for audio files and generate Blaise code to play them.

3.2 Monitoring Changes

A key feature of QSD is its ability to track all changes made to the specifications. Every time a change is saved, information is recorded about who made the change, the date and time, and the kind of change made. By selecting the button “Changes to Specs by Questions” from the “Select Module” screen, users can filter, sort, and prepare custom change reports.

This is very helpful for translators to check for updates after revisions are made by the spec writers. Programmers use it to determine which modules need fields updates, or there is a need to re-populate types and/or fills.

Central logging of changes and change requests streamlines communication, reducing the confusion and wasted time characteristic of poorly coordinated email exchanges.

It also allows monitoring the history of all changes that were made to any given variable of the questionnaire. By looking at the corresponding version of the variable in Microsoft Visual Source Safe, we can tell exactly what was changed. While all previous versions could be saved within QSD, we prefer to require the use of source code control for our instruments.

3.3 Monitoring Problems in Testing

During the testing phase, testers and programmers use QSD to log problems as they are discovered. After a problem is corrected, a programmer marks the problem as fixed. When the whole module is updated, the programmer records this using the button “Verifying Changes to Blaise Code” from the “Select Module” screen. QSD has reports which show the list of modules that need verification by programmers and by testers. Before a new version of the instrument is released for testing, all modules are verified by programmers. When many programmers are working on the instrument, this report helps the build manager to be aware of the statuses of modules. Testers note the test outcomes for each module. All modules are verified as having passed testing before being released into production.

3.4 Producing Targeted Specifications

Since QSD is the single repository for the specification from which the code is generated, it is not necessary to check consistency of wording, question order, and so on, between specifications and program. It is very convenient for testers to do their jobs and for questionnaire designers and clients to review specifications at any point of the development process, with confidence that what they are viewing is in step with the current instrument.

Figure 5. Specification documents show pre- and post-logic, label, tag, fills, and response options.

```
-----
Logic before:
IF GI_work=yes then ask GI_org
GI_org [ GI2 ] how often appreciated by people or leader
How often ^Fills.GI_org_1 by the leaders or people of your organisation for the work and activities
^Fills.GI_org_2 at your organisation?
Question Type: TNotSome/veryAll
NotAtAll 1 Not at all
Sometimes 2 Sometimes
Often 3 Often
VeryOften 4 Very often
AllTheTime 5 All the time
Question Fill: ^FillGI_org_1
present do you feel you are appreciated
past did you feel you were appreciated
Question Fill: ^FillGI_org_2
present that you do
past that you did
Logic after:
IF GI_org = notAtAll skip to GI_why
```

QSD produces a specification document in a format that shows all elements of each field. Specifications can be printed in English only, Spanish only, or both languages at once for a selected module or an entire instrument. They may be exported in any format supported by Microsoft Access.

4. Generating and Updating Blaise Instruments from QSD

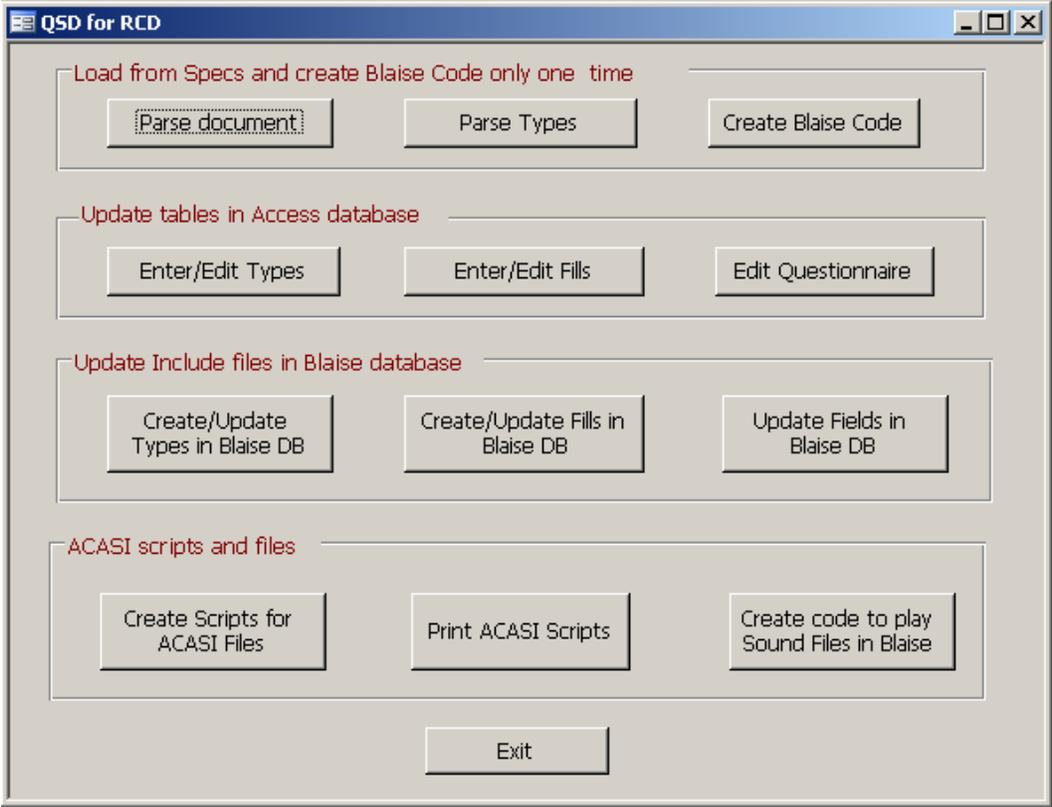
Blaise promotes modular questionnaire structure. A modular structure delivers efficiencies in code maintenance, division of labor, and data model preparation (compilation). In addition to these advantages, a modular structure facilitates the code generating and updating capabilities of a tool such as QSD.

Each section in our Blaise instrument is programmed as a separate block. All these blocks are incorporated into the instrument as included files. All types used in the instrument are compiled into one INCLUDE file. Fills declarations and their types are also separate INCLUDE files.

Since Blaise allows the separation of fields and rules at the block level as well, each block has an INCLUDE file which contains all fields associated with the specific block.

This important feature of Blaise allows dividing responsibilities between programmers and survey specialists.

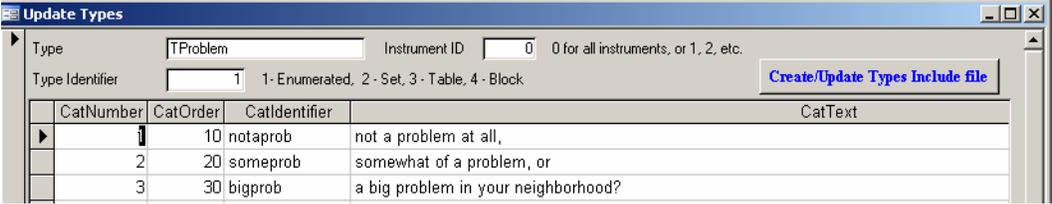
Figure 6. The interface for programmers reflects the distribution of the Blaise instrument's various elements among different files.



QSD is capable of simultaneously loading all user-defined types needed by the Blaise instrument. In order to do this, the programmer uses the “Parse Types” option and selects a properly formatted Microsoft Word document.

Types may also be added or edited at any point during the development process by using the “Enter/Edit Types” option.

Figure 7. Types and fills can be edited and quickly pushed to the Blaise instrument.



The same is applicable to fills, using an interface similar to the one shown. The programmer’s goal is to use an approach where there is no hard coded text in the Blaise instrument.

At any point in the life cycle, the programmer can update types, fills, and fields for the whole instrument.

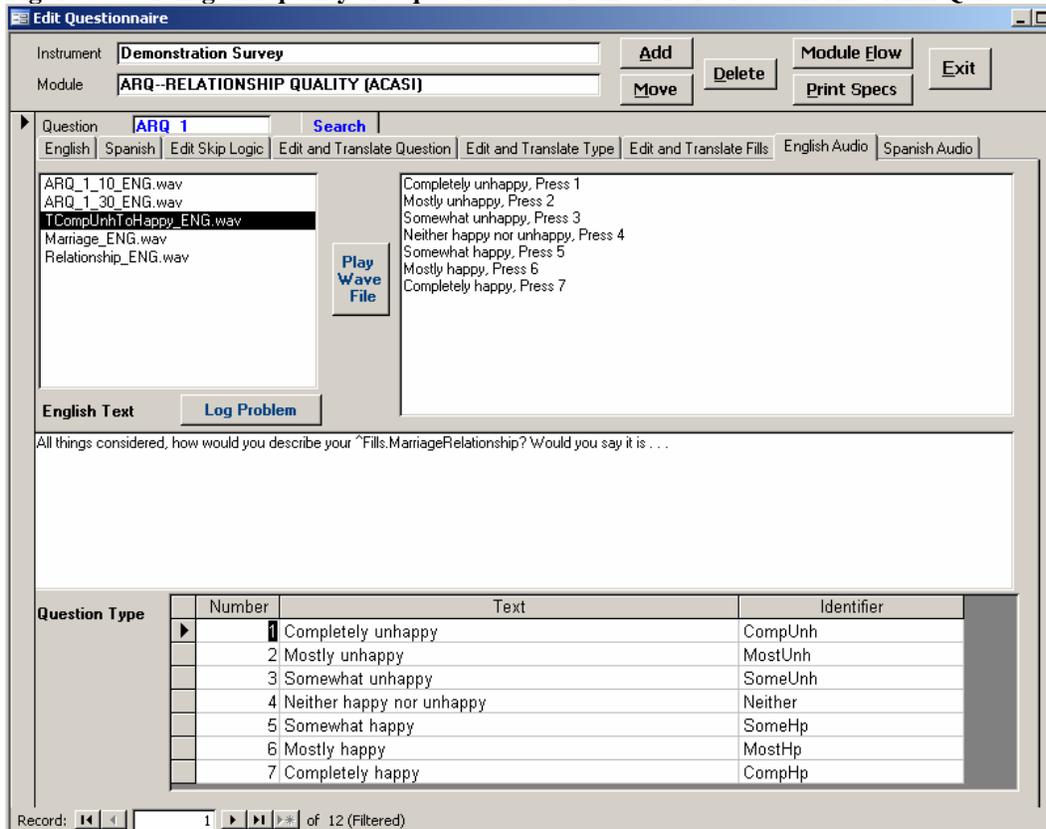
5. QSD and ACASI

For studies that require use of audio files, QSD helps to produce scripts for recording. Each question that is to be recorded is marked in QSD. Marked questions cause additional interface elements to be visible on the “Edit Questionnaire” form, and an entry for an additional language to be added to the Blaise code.

5.1 Testing Audio Files in QSD

Due to fills and types with response options, there are almost always several audio files for each question. These need to be tested carefully to provide the respondent a smooth listening experience and to ensure audio coverage of fills under various scenarios. It is time consuming to do such testing solely through interactive use of the Blaise instrument. Testers can use QSD to check that all necessary files are recorded for the question, that the pronunciation is good, and there is smooth transition from one file to the next.

Figure 8. Wording and quality of a question’s audio files are reviewed from within QSD.



Testers log any problems with audio files into QSD. Programmers then update the scripts for re-recording.

5.2 Implementation in the Blaise Instrument

Some studies require that the respondent have a choice to listen to the questions only without seeing the question text on the screen. We satisfy this requirement by adding a so-called “Private” language to the Blaise instrument for each natural language used. If

the instrument has two languages, two “Private” languages are added. To maintain the multiple languages, we use an auxiliary field LNG.

Figure 9. In this example of QSD-generated code, the value of an auxiliary field W_AC_2 holds information about audio files associated with the question AC_2.

```

{*****}
{*****Rules**FOR**ACASI*****}
{*****}
    W_AC_1 :=
        ' SOUND('+ WaveDir + '\AC_1_10_' + LNG + '.wav)'
    W_AC_2 :=
        ' SOUND('+ WaveDir + '\AC_2_10_' + LNG + '.wav)'
        + ' SOUND('+ WaveDir + '\TMonth_' + LNG + '.wav)'

```

Languages are defined for an entire Blaise instrument, so we cannot direct some modules to use a “Private” language and others to not use it. Thus, for questions that lack sound files, text for the “Private” language is generated from QSD with the same text as the corresponding natural language. This way the correct text is always presented on the screen. For questions where only audio should be played, the text is replaced by the special character ".". Blaise will recognize it as space to be displayed on the screen.

Figure 10. Audio-only questions generate a specially formatted field definition.

```

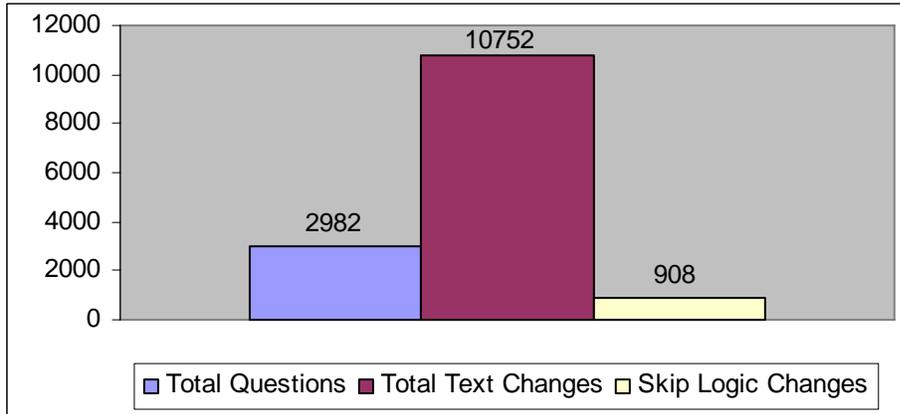
AC_2
ENG
    "In what month were you born?"
SPN
    "¿En qué mes nació usted?"
MML "^W_AC_2"
PRV "...."
PSP "...."
/"Month of birth (practice)"
: TMonth ,noDK, noRF

```

6. QSD at RTI International

In using QSD on a number of studies over the last two years, we at RTI International have found that it reduces questionnaire development time. The number of questions maintained in QSD has exceeded 9000 among 26 instruments across several studies. As Figure 11 shows, most changes are to text, and QSD gives us the flexibility to accommodate a high volume of these changes in a short period. On one recent study, 442 changes – mostly to wording – were logged in a single day late in the development cycle. A new version was ready for testing that evening.

Figure 11. QSD log data from four recent bilingual studies conducted by RTI International illustrate that great opportunities for efficiency lie in accelerating text changes.



We continue to add new features to QSD and improve the existing ones to satisfy requests from users. Future plans include a web interface and the option to use SQL Server for the back end.

7. Conclusion

QSD is a central place to enter and refine questionnaires, from organizing sections to adding and changing questions and logic, to keying translations, to associating audio files for ACASI. It has built-in support for problem tracking, change tracking, and version control.

Advantages of QSD include:

- Specs-Instrument synchronization;
- Programmer focus on logic rather than question wording;
- Addition of a second language without programmer intervention;
- Extensive support for ACASI.

QSD helps speed up development and simplify maintenance of questionnaires. It promotes the creation of error-free Blaise instruments.

Since internally QSD stores all questionnaire elements in a relational database, alternate code generation modules could be developed and plugged in so that the same definition could be used to create a questionnaire in a language other than Blaise, or to take advantage of new features in future versions of Blaise.

8. Acknowledgements

The authors would like to thank members of the Research Computing Division and Survey Research Division at RTI International for their use of QSD, as well as for their invaluable comments and suggestions related to its further improvement.

Authoring Blaise questionnaires – a task for the survey specialist or an IT programmer?

Rebecca Gatward, Office for National Statistics, UK

1. Introduction

All organisations approach the development of Blaise questionnaires using their own unique process. We know that their approaches, however, can be broadly divided into two models. In the first, a dedicated team of Blaise programmers develops the questionnaire based on specifications produced by the survey specialist. In the second, survey specialists develop the Blaise questionnaire themselves. We also know that the first approach is the dominant one amongst Blaise-user organisations.

Kinsey and Jewell (1998) confirm this as they state that, ‘Very different strategies and methods are being used successfully in different organisations. Perhaps the most fundamental difference among firms in CAI development is whether the same or different people are responsible for content development and CAI’.

The aim of this paper is not to argue that one of these approaches is the ideal model but to review the existing approaches, based on information obtained from Blaise user organisations. The review will include a summary of the models, a discussion of the advantages and disadvantages of each, explore why approaches differ between organisations, and draw some conclusions about the efficiency of the models.

The discussion in this paper may be useful to an organisation that is about to start using Blaise by helping it make the decision about which broad model it will use. Similarly existing users may find there are aspects of another organisation’s approach that would be useful to adopt in their own organisation.

2. Background

Most Blaise-user organisations have been using the package for between 5 and 20 years, and therefore now would be an appropriate time to review the two broad approaches. One of the aims of this paper is to assess how the approaches organisations adopted have changed over time.

Previous discussions on approaches to authoring CAI questionnaires have covered descriptions of the various models or concentrated on particular stages of the authoring process, for example, testing or specification. They were also written at a time when the move from PAPI was still fairly recent and so included discussions about the transfer from PAPI questionnaires to CAI and the implications of this transition. What has been explored to a lesser degree is why decisions were made to adopt a particular approach.

Clark, Martin and Bates (1998) state that, ‘In an automated environment one confronts a need for instrument authors namely, persons knowledgeable in CAI programming languages’.

There are, however, very few guidelines on how to approach authoring Blaise questionnaires and who the instrument authors should be. Blaise developers provide the

following statement in the Blaise manual, in a section outlining the benefits of the package,

‘Subject matter specialists, statisticians, and programmers can become adept at authoring Blaise instruments. The modular and reusable structure of the language allows many surveys to use the same blocks of code with little or no modification. This results in faster, surer development and better comparability between surveys. The multi-mode nature of Blaise encourages (and can enforce) consistent specifications and conventions between multiple modes of use.’
(Blaise Developers Guide)

To get the best out of Blaise, the Blaise team recommends that for using the basic part of the system, a social scientist, who is working as survey designer and who has at least some experience in IT and statistical methodology is well equipped to create and implement a Blaise survey, and is therefore suitable for this job.

Although the Blaise team do acknowledge that the Blaise system has been growing, enabling to create more powerful and more advanced applications. When organisations want to get more out of the system, one would need programmers that can handle these new and often complex parts of the system, such as the Blaise Component Pack, Datalink and Blaise IS. Also the number of options inside the languages Blaise and Manipula have been growing. For specific projects that need expertise of Blaise at a higher level, using the complex parts of the system, one may ask for programmers with a higher level of education and or experience in IT.

3. Blaise questionnaire authoring across user organisations

3.1 About the organisations

To gather information on current authoring practices, a short questionnaire was sent to 34 Blaise user organisations, the following 11 responded.

- Statistics Canada
- Statistics Slovenia
- Hungarian Central Statistical Office for National Statistics
- Statistics Belgium
- Survey Research Center, Institute for Social Research, The University of Michigan
- Australian Bureau of Statistics (ABS)
- Istituto Nazionale di Statistica (ISTAT) , Italy
- WESTAT
- Statistics Germany
- Statistics Netherlands
- Office for National Statistics, UK

A copy of the questionnaire is provided at appendix A.

These 11 organisations provide a good range in geographic location (i.e. Europe, USA, Canada and Australia), size and experience using Blaise (long established and more recent users, between 6 to 20 years). The responding organisations provide a mix of type of organisation although the majority are National Statistical Institutes.

The use of alternative CAI software prior to using Blaise also varies by organisation. The majority of European organisations had not used any other CAI software prior to Blaise, unlike the American and Canadian organisations who had – such as CASES, Surveycraft and Cheshire (WESTAT’s in house CAI software development system).

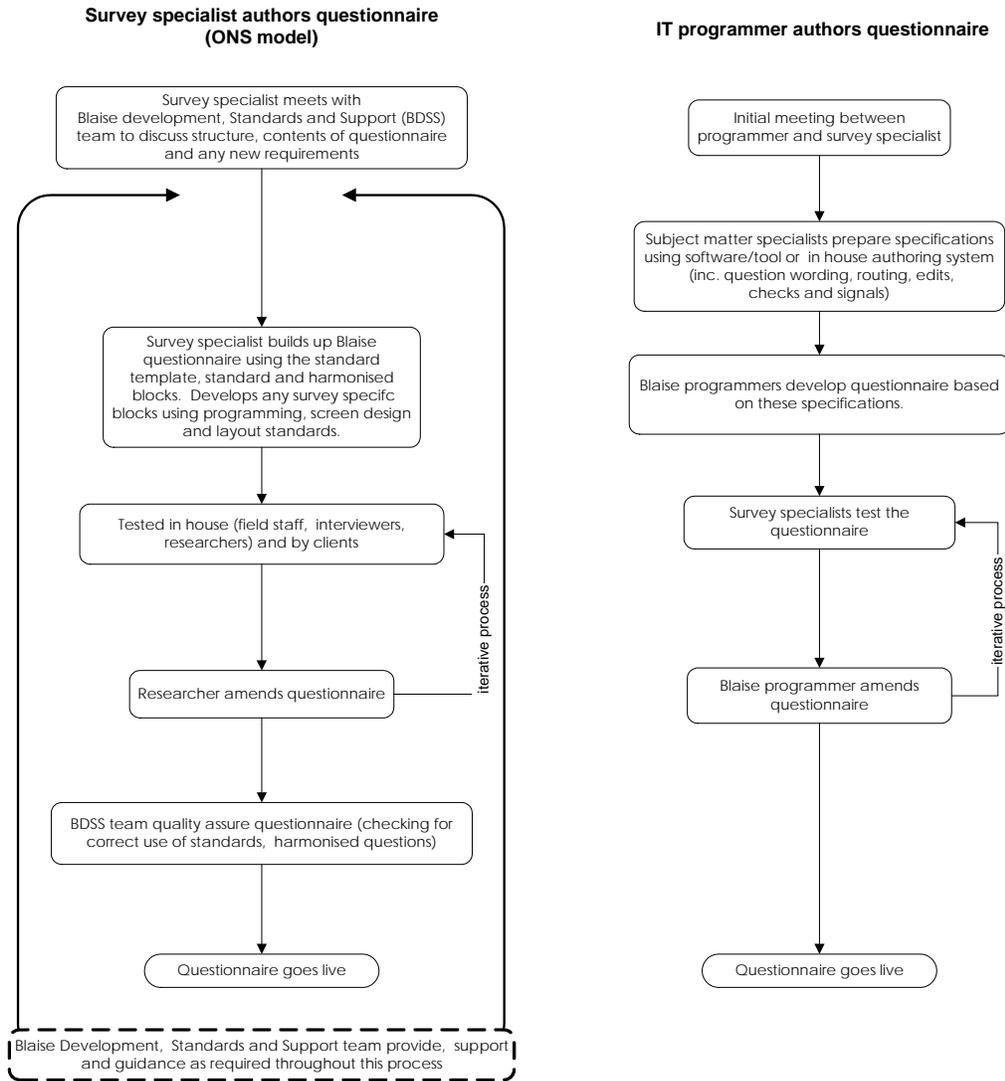
Blaise is used by the organisations to undertake a variety of survey types, including, social, business, institutional and agricultural also using a range of modes of data collection, the most common being CATI and CAPI. The extent to which Blaise is used along the survey process also varies by organisation; all use certain components such as the Data Entry Program (DEP) and Manipula but not all use others, such as Blaise IS or Bascula.

3.2 Summary of approaches

Information provided by the organisations confirms what we would expect, that is, that only two broad approaches exist and that the IT programmer approach is the dominant one.

The following chart describes these two main approaches, highlighting the common stages in each approach and not the points where some organisations deviate from this model. As ONS is the only organisation (from those who responded) which use the survey-specialist model that part of the chart is based on the ONS approach.

Chart 1. Summary of approaches to authoring Blaise questionnaires



3.2.1 Survey specialist authors Blaise questionnaires

ONS is one of the few organisations, and the only one amongst this sample of organisations, where the survey specialist (survey researcher) is responsible for authoring the entire Blaise questionnaire. Colleagues in IM are responsible for case management systems, telecommunications and data output.

Although ONS is the only organisation, of those who responded, Statistics Netherlands have used the approach in the past. When they started using Blaise in 1987, the designer of the questionnaire was the same person as the developer of the questionnaire in Blaise. At this stage the Labour Force Survey was the only survey in Blaise. The number of surveys using Blaise increased and more people working on survey design and implementation resulting in division of staff into specialisms, such as survey designers and Blaise programmers. Now the survey designers write specifications (using VISIO) and then pass them to the Blaise programmer to implement the survey in Blaise.

In order to understand how the process works within ONS, it is important to explain the key elements that are in place to make this approach work.

It is helpful first to describe the role of the researcher within ONS. Researchers are social scientists with some statistical expertise; the entry requirement is a good degree but in reality most have a Masters degree or higher. They are survey generalists i.e. they move from project to project to ensure they receive experience of different type of survey and subject matter. Researchers manage surveys as projects through matrix management of teams and so requires knowledge and control of all aspects of projects (client liaison, sampling, questionnaire design and analysis).

Key to the success of this approach are standards, training and support.

- **Standards**

ONS has recognised the importance of standards since starting to use Blaise for CAI. The benefits of using standards are well documented and are evident along the entire survey process from authoring a questionnaire to data quality.

At ONS, Researchers use the following standards which assist the questionnaire design and authoring process.

- Standards for writing Blaise code, for example, naming conventions and layout of code.
- Screen design and layout standards.
- ONS also maintains standard Blaise code – including blocks of harmonised questions, standard code, for example, Standard Occupation Coding and relationship grid and templates which define the structure of the questionnaire for example, the datamodel rules block and .BLA file.

A further standard or principle is that Blaise code is reused whenever possible and to keep it simple and not overcomplicate code for the sake of it.

- **Support**

Key to the success of ONS's approach is the provision of sufficient support for the researchers who are authoring Blaise questionnaires. More experienced Blaise programmers support those with less developed skills. In the past this was an informal arrangement, and to some extent remains so.

The provision of support was formalised with the creation of the Standards, Quality and Assurance (SQA) team, comprising people from Field, IM and Research branches who had a particular interest in undertaking development projects or those with particular skill to provide support to others. The team members had a limited amount of time to spend on this work and had to fit it in alongside main project work. Fitting the work in became increasingly difficult and development projects progressed very slowly or not at all. To overcome this problem, a smaller team was formed with members having dedicated time. The team, renamed Blaise Development, Standards and Support (BDSS) team, is currently made up of two researchers who spent half their time on development work and the remainder on survey project work and two full time members who concentrated on standards and support work, although in practice they are also involved in development work. A fifth member would be nominated from the

Information Management (IM) section who, it is intended, promotes the use of standards amongst their colleagues in IM, advises on the more technical aspects of Blaise and its implementation and carries out development projects.

The BDSS team,

- holds an initial meeting with author to agree the design of the questionnaire
 - provides support to authors throughout the process
 - reviews and promotes the use of standards
 - maintains standard blocks of questions
 - carries out development work
 - passes on knowledge of new capabilities.
- **Training**

All researchers are expected to learn and be able to program in Blaise; senior managers need to understand enough of the critical design issues to be able to supervise junior staff who are authoring the questionnaires. A half-day introduction to Blaise is sufficient to provide an appreciation of Blaise. Those who will be authoring Blaise questionnaires attend this half-day course and also a three-day course provided by Statistics Netherlands.

ONS is fairly unique in its approach to Blaise questionnaire authoring, so why did it decide to use this approach?

ONS has always regarded CAI software as a tool for researchers rather than a tool for IT specialists. Manners (1998) suggests that as PCs spread in the workplace and PC software became easier to use, researchers became more accustomed to hands-on control in areas of the survey process which they may not have had in former years. It was also felt that Blaise software required no greater computing skills than their researcher already exercised in using statistical analysis packages, like SPSS for manipulating data (using logic as well as just running tables). CAI therefore appeared a natural progression for researchers in their work with computers.

ONS continues to believe that the questionnaire design should be translated directly by the researcher, who therefore constructs the questionnaire using Blaise. In PAPI days, this involved working with scissors and glue; in CAI this process was carried out by cutting and pasting electronically.

A management review confirmed the policy that researchers should write the Blaise questionnaire and recognised the following considerations: (Manners and Green 1996)

- Importance for survey quality of hands-on control and knowledge of the questionnaire by researchers.
- Stressed the importance of co-operation in the survey team between researchers and computing specialists to ensure that the requirements for efficient output design were built into the datamodel by the researchers from the start.

- A model which required researchers to write specifications for programmers to translate into Blaise is duplication of resource and potentially error-prone.
- Researchers will spend a small and intermittent part of their careers writing Blaise questionnaires. This meant that there was a need for need for a cost-effective mechanism to ensure researchers had the necessary up-to-date skills at the point they were required.

This approach has since been critically reviewed several times since ONS started using Blaise in 1990 (for production), and has always been confirmed as the best way to meet ONS's goals in survey design.

Recently the effectiveness of this approach has once again been examined. ONS is currently going through a period of major organisational change which is testing the ONS model. Social and Vital Statistics Division (which contains the Researchers and BDSS) is relocating out of the London office to the ONS office in Newport (now ONS HQ). This has resulted in almost an entire Division of newly recruited staff who have no previous Blaise experience. This has had an impact on the work of the BDSS team, they are having to spend more time providing support, training and in some cases writing sections of Blaise questionnaires and less time working on development projects.

Additional factors that are impacting on the approach are (1) the changing profile of the survey work ONS is currently undertaking which means there is less opportunity to build up expertise in Blaise and (2) the transition of a major ONS survey from cross sectional to a longitudinal survey (detailed in Setchfield 2007).

During this period of change ONS has drifted from the survey specialist taking responsibility for all questionnaire authoring to more of a hybrid approach, as described below. It is hoped that this is a temporary change. It is early days, and ONS is still convinced that this approach to Blaise questionnaire authoring is the most appropriate method.

Hybrid model

A further model that can be included under this broad approach is the Hybrid model. From knowledge, this model is used, to varying degrees, by a small number of organisations. It could be argued that, due to the factors explained previously, ONS has moved closer to this model over the two years.

Before describing this model it is important define the Blaise expert. A Blaise expert is someone who has used built up an expertise and is not necessarily an IT specialist, although they could be.

In this hybrid approach authoring a Blaise questionnaire is not the responsibility of one individual or one functional team in an organisation. Instead the responsibility for Blaise programming is divided between the survey specialist and the Blaise experts. The division of responsibilities may vary between organisations. For example, the survey specialist may be responsible for programming the FIELDS (questions) and the simple routing i.e. the 'research' parts, and the IT programmer completes the more complex parts of the questionnaire such as the concurrent interviewing or rotation of data for dependent interviewing.

3.2.1 IT programmer authors questionnaire

The majority of the organisations who responded used the IM programmer model. In this approach, questionnaires are developed by a team of Blaise programmers. The survey specialist provides a specification by hand or using authoring software which is then passed to the Blaise programmer who develops the questionnaire based on the specification.

The organisations which use this model were asked to define what qualifications they have or their background. Most stated that their IT specialists had Computer science degrees or had a knowledge of a range of software packages.

Organisations who follow the IT programmer model broadly follow the same approach but they deviate from the process in their own unique models. Here are some examples,

- Statistics Canada tests questionnaires at Block level, once these are signed off the front and back end of the questionnaire are then tested; for more complex instruments the blocks are integrated and tested prior to adding the front and back end.
- ISTAT have a team of Blaise experts who develop all questionnaires, alongside the survey specialists, except the Labour Force Survey (LFS). As the LFS is a continuous survey, changes are made by a different team of IT programmers.
- At Westat the specification process is facilitated using an in-house authoring system called Specwriter which is used by both survey specialists and Blaise programmers.
- ABS also have a Survey development tool. This currently produces specifications for a Blaise questionnaire instrument, and they hope to extend the functionality so the software also generates Blaise code.

Approach can also vary by type of survey - for example, at ABS, IT specialists, who are responsible for more than just Blaise programming, develop business surveys, whereas social survey questionnaires are developed by specialist Blaise programmers (although in the near future the two groups will be merged).

There were a variety of responses as to the why their organisation decided to use this model. The most common being that it fitted in with their organisational structure, specifically the traditional division of labour between statisticians and computer specialist. For another organisation, the decision was made due to practical experience and because this method was shown to work. Others chose the model owing to having the advantages of having a central pool of Blaise programmers who use the software continuously and so become more skilled in the software.

ABS said that, while there is no objection to the suggestion that development staff could write their own instruments, there is a feeling at the ABS that people with training and technical skills in question design and testing are better to focus on questionnaire development work, while others with skills in programming are better to focus on the activity of Blaise programming.

Most European-user organisations and ABS had no experience of CAI software before starting to use Blaise. Amongst North American organisations, the majority had used on

of a variety of CAI software packages prior to Blaise, the most prominent being CASES. As the IT specialist is the dominant model amongst all user organisations we can conclude that use of previous software, which may require more IT skills has influenced the approach they use to Blaise authoring. In addition, given the relatively high staff turnover in some organisations survey infrastructure areas it would be difficult for someone to become highly proficient in both areas of work. One organisation using this approach highlight the creative tension between the question designers and Blaise programmers that has been a healthy one for the surveys and pushed the limits of what is possible.

4. Discussion

The following section will outline the perceived advantages and disadvantages of the two broad approaches to Blaise questionnaire authoring.

4.1 Survey specialist authors Blaise questionnaires

Advantages of this approach:

- Blaise skills and expertise are widely spread amongst staff, an organisation does not need to depend on a few key staff with Blaise programming skills. Although the variation in skill levels does mean additional support is required to tackle more complex sections.
- Researchers develop a knowledge of the capabilities of Blaise and can use them to make improvements to current survey or future surveys and also in discussions about questionnaire design with clients.
- Blaise language is similar to SPSS syntax language (the statistical software mostly used by researchers) so they should be able to adapt their SPSS skills relatively easily to use Blaise. However, more recent researchers use windows, reusing existing syntax now, so there may be a longer learning curve for Blaise.
- Blaise software is intuitive and so precludes the need to write an English specification and have that translated by experts into a computer program, this means there is one less stage and eliminates the risk of errors caused by poor specification.
- Researchers design the questionnaire as they program, after some initial planning, they know the subject area, aims of the project and can spot illogical routeing.
- Researcher gets to know the questionnaire very well. They are able then to use this knowledge when briefing interviewers, writing the questionnaire instructions, dealing with questionnaire queries and carrying out the data analysis.
- The researcher maintains control over the whole questionnaire design process i.e. question wording, screen layout, on screen help, routeing, checks, computations and outputs.

Disadvantages of this approach:

- Researchers use Blaise intermittently and so have to relearn each time they start a new project.
- Writing Blaise questionnaires is not perceived by some to be an appropriate task for researchers. Some researchers view the task as mundane.
- Do not have a specification to test the questionnaire against. Although, in practice, a paper questionnaire is often created by the clients or researchers either from scratch or from existing questionnaires at the beginning of the project.
- Training is difficult to plan – there are not always sufficient researchers starting at the same time to fill a course.
- Blaise authoring can get designated only to one team member.

4.1.2 Hybrid model - Blaise questionnaire authoring is divided between the survey specialist and Blaise experts

Advantages:

- Structure of the questionnaire is designed for maximum efficiency of data output, readability or reusability, incorporates standards.
- Allows the researchers to concentrate on question construction, screen layout and testing.
- Experts have experience of working on surveys so they do not ‘lose touch’ with reality.
- Experts responsive to research needs and familiar with Blaise so an ideal position to exploit new possibilities.

Disadvantages:

- Timetabling can be complex, availability of Blaise experts may be problematic as resource must be shared across surveys. Work may also be sporadic, for example, during the preparation for the new survey year on the continuous surveys. Although this may allow time to work on development projects during the ‘quieter’ months.
- Relies on good communication and close working between the teams.
- Susceptible to organisational change – reliance on expertise of a few.

4.2 IT programmer authors questionnaire

Advantages:

- IM staff have the programming and logic skills to program a Blaise questionnaire quickly and efficiently.

- IM staff have the skills to carry out the systematic testing required of the program.
- Training required is minimal.
- Blaise programmers are more skilled in Blaise because they work with it continuously.

Disadvantages:

- Relies on close working of IT and Research members of the team. The IT specialist may not necessarily have the social research knowledge to make informed decisions or understand the context of the work.
- Specification process results in wasted time and introduces errors.
- IM staff may not feel that Blaise programming is sufficiently ‘challenging’.

5. Conclusions

All organisations responded positively when asked how well they felt their approach to Blaise questionnaire worked. It would seem organisations have built up their own unique models that are now embedded within the organisation and work well for them.

There are, of course, perceived advantages and disadvantages of both approaches and some organisations highlighted changes they would like to implement to improve the process. However, since organisations chose to proceed with a particular approach they have spent time making it work within their organisation. From the information supplied by the 11 organisations it is evident that only one organisation had swapped from using survey specialists to IT programmers authoring Blaise questionnaires. After using Blaise software for up to 20 years I would predict that user organisations will now continue using the same model and refining it as necessary.

I would also suggest that it is necessary to regularly evaluate how effective a model works within an organisation. Examples provided in this paper do demonstrate that factors, such as organisational change, can alter the effectiveness of an approach and lead to temporary or permanent changes. This implies that organisations need to be flexible and be willing to adjust their approach when necessary.

From the information provided by the 11 organisations, is not possible to conclude that one approach to Blaise questionnaire authoring is more efficient than another. This reaffirms that no single CAI development method works for every organisation or survey (Kinsey and Jewell 1998). Although it has enabled an updated description of how a selection of Blaise user organisations go about authoring Blaise questionnaires.

This all implies there is no optimum or ideal approach – all that does matter in terms of authoring is that the person authoring the Blaise questionnaire has sufficient Blaise expertise, adequate support and a willingness to learn.

6. References

Kinsey S.H., and Jewell D.D. (1998) A systematic approach to instrument development, Chapter 6 of *Computer Assisted Survey Information Collection (Edited by M.P. Couper et al)*, pp.105 - 124.

Clark, C.Z.F., Martin, J., and Bates, N. Development and implementation of CASIC in Government Statistical Agencies, Chapter 4 of *Computer Assisted Survey Information Collection (Edited by M.P. Couper et al)*, pp.63 - 84.

Statistics Netherlands (1999), Blaise Developers Guide, Blaise for Windows 4.1 A Survey Processing System

Manners, T. Using Blaise in a survey organisation where researchers write the Blaise datamodels Proceedings of the International Blaise Users Conference, 1998 Lillehammer, Norway 1998

Manners, T. and Green, H. (1996): Blaise III: Who should do what? Internal Social Survey Division (ONS) paper, July 1996

Setchfield, C. Coping with people who just won't stay put: The Use of Blaise in Longitudinal Panel. To be presented at the 11th International Blaise Users Conference, 2007, Annapolis, September

7. Appendices

Appendix A

Blaise authoring questionnaire

Name of organisation

Your name and position within organisation

Please provide a brief description of the scope of work undertaken by your organisation.

Date/year when your organisation start using Blaise

Version number of Blaise you used for the first 'live' questionnaire

Did you use a different CAI package prior to Blaise? If yes, then which one

Which version(s) are you currently using for 'live' interviewing?

Current authoring practice. Please describe the process from question development and negotiations with clients up to the Blaise questionnaires 'going live', including details of the specification and testing stages

Why did your organisation decide to use this approach to CAI authoring?

Has the current approach evolved over time? If so, how and why?

If a dedicated team of Blaise programmers author the questionnaires - what are their backgrounds/qualifications?

Which elements of Blaise do you use within your organisation? (i.e. CATI, Bascula etc.)

How do you deal with 'in year' changes to the Blaise questionnaires?

If possible, it would be useful to have an estimate of the length of time it takes your organisation to program a Blaise questionnaire. Or perhaps you could provide an example timetable for a recent survey.

Again, if possible, it would be useful to have an indication of the level of accuracy of your Blaise questionnaires. For example, frequency of re-issuing questionnaires to interviewers.

How do you feel the process works? - are there any changes you would like to make?

Blaise Source Code Editing System

Sheila Deskins and Danilo Gutierrez, The University of Michigan

1. Introduction

This paper discusses a Blaise Source Code Editing System designed and developed by Health and Retirement Study (HRS) programmers. It covers the six major functional design components of a source editing system, as well as specifics relating to application design, development, and testing plans that allowed the creation of the current working system.

The Blaise Source Code Editing System or “Source Editor” is used to make several hundreds of updates to the Blaise Source files (.bla/.inc) automatically. In the past the process of updating information from electronic review systems had been incorporated into Blaise code by hand, which was a labor-intensive, tedious and error prone process.

The six major functional design components are: a file reader/parser to process the Blaise source code files, a maker of expanded Blaise statements, a translator/pre-processor to change electronic update information into a format the merger can process, a merger that combines the update information and the Blaise source code, a file writer to produce the revised files, and an interface to encompass the whole system.

This paper will go through the evolution, design and current state of the Source Editor System. Topics covered are: the motivation behind having such a system, key design concepts, the implementation of the modular design, the testing plan for systems development, and current and future uses of the system. An appendix, containing a sample program with details of tables used as I/O in key design components of the system, is included.

2. Background

The Health and Retirement Study (HRS) is a longitudinal study that began in 1992. For the 2002 data collection period, HRS’s CAI¹ application switched to Blaise. Although the basic questionnaire content of a longitudinal study does not change over time, there are many changes in business practices. Changes such as implementing revised screen formatting guidelines and revisions which facilitate post data collection distribution are typical.

Changing needs such as these often translate into frequent large scale changes to HRS’s CAI application. The number of lines of code requiring modification is usually counted in the hundreds or thousands. These voluminous changes are programmed into the CAI prior to each field period. The work is labor intensive and time consuming. Thus HRS wants to automate this work as much as possible.

Development of the source editor system began in 2003 when a programmer was given an electronic file with 2,700 descriptors to update. The time frame allotted for the task was based on doing the updates by hand. A source editor system was created which

¹ Computer Assisted Interview

consisted of a parsing application writing to a database and a series of ad hoc merging and writing routines. Descriptors were done using this early system. Later, a screen format change resulted in a few hundred fields needing updating. The updating was also done via this early source editor.

For the 2006 CAI application a few thousand descriptors were updated. The descriptor input file had a different format which required new ad hoc routines to be created. One routine was created to translate Blaise Data Entry Program (DEP) names to defined field and block names. Another routine identified duplicate update requests based on the mapping of the DEP name to a defined field and block name.

It was decided for the 2008 CAI to add a new language. This very large task prompted the programming group to develop the current system. The system needed to handle more than just field text and descriptor updates. With the complexities and the larger scale of the new language task, the system needed to become more robust and required additional functionality. This was the first large scale task identified for 2008 and more changes are planned.

3. Design Conceptualization

What is a source editor? A source editor is a set of related processes and applications that allow for the editing of Blaise source code files. Blaise source code files are files with .bla and .inc extensions that contain Blaise source code to compile and generate a .bdb, .bmi, and other files.

The source editor system preserves the original .bla and .inc file structure. The basic core components of the source editor are the parser, merger and writer. To edit the source code, we have to have the original source files available, a collection of planned changes (or updates) and a way to save and write the updated changes. The source editor system is designed to be a bulk editor. That is, to make several hundred changes at once.

When talking about editing Blaise source code files, what do we mean, exactly? Do we get to add new fields, new question text, new enumeration code names, new enumeration code labels, or new user defined data types? Do we get to add blank lines, whitespace, and comments? Can we modify what is there? How much can we delete? What assumptions are being made? Answering these questions helps to provide insight into the source editor system design and conceptualization.

Beginning with the first question, “Do we get to add new fields?” the answer is no. The goal is to edit or update existing Blaise source code, not to add new². The Blaise source code consists of existing Blaise program statements. Our bulk updates deal with changes to text and languages. This begs the question “Is adding a new language something the source editor can do?” Adding a new language is not really adding something new. If you examine a Blaise statement for the field syntax, the statement allows for all languages defined by the languages statement.

² Adding new statements implies development of an authoring system, a much different concept than an editing system. HRS has a working authoring system which is not covered in this paper.

Syntax
FIELDS

Q [Q1, [...]] [(Tag)] [[Lid] "Text"] [...]
[/ [Lid] "Description"] [...] : T

All parts of the statement syntax may not be explicitly coded. However the statement does exist. It is desirable for the source editor to update all parts of the statement, especially since many of the bulk changes deal with languages. For example, if two languages are defined and the statement is coded as:

Fields

Q1 (Q1) "Are you ready to answer questions?":(y,n)

There is implicit room in the above statement for a second language text, two descriptors, and language identifiers for all the text. Table 1 below shows the statement as parsed, with explicit token types and corresponding tokens. Table 2 below shows the expanded statement with the implicit token types added.

Table 1 Statement as Parsed

LnNoBeg	ColBeg	LnNoEnd	ColEnd	Token ³	TokenType
170	2	170	3	Q1	FName
170	4	170	4		WhiteSpaceBlank
170	5	170	8	(Q1)	FTag
170	9	170	9		WhiteSpaceBlank
170	10	170	45	"Are you ready to answer questions?"	Text
170	46	170	46	:	FDefnSep Separator :
170	47	170	47	(EnumBeg Separator (
170	48	170	48	y	EnumCodeName
170	49	170	49	,	EnumSepSeparator ,
170	50	170	50	n	EnumCodeName
170	51	170	51)	EnumEndSep Separator)

Table 2 Statement as Expanded

Line NumBeg	Column Num Beg	Line Num End	Column Num End	Token	TokenType	LangID	Code Value
170	2	170	3	Q1	FieldName		
170	4	170	4		WhiteSpaceBlank		
170	5	170	8	(Q1)	Tag		
170	9	170	9		WhiteSpaceBlank		
	9				FieldTextLangID	ENG	
170	10	170	45	"Are you ready to answer	FieldText	ENG	

³A token is part of a program statement consisting of characters identified as meaningful syntax.

Line NumBeg	Column Num Beg	Line Num End	Column Num End	Token	TokenType	LangID	Code Value
				questions?"			
	10				FieldTextLangID	SPN	
	10				FieldText	SPN	
	10				Separator/		
	10				DescTextLangID	ENG	
	10				FieldDescriptor	ENG	
	10				DescTextLangID	SPN	
	10				FieldDescriptor	SPN	
170	46	170	46	:	Separator:		
170	47	170	47	(Separator(_Enum		
170	48	170	48	y	CodeName		y
	48				CodeTextLangID	ENG	y
	48				CodeText	ENG	y
	48				CodeTextLangID	SPN	y
	48				CodeText	SPN	y
170	49	170	49	,	Separator,		y
170	50	170	50	n	CodeName		n
	50				CodeTextLangID	ENG	n
	50				CodeText	ENG	n
	50				CodeTextLangID	SPN	n
	50				CodeText	SPN	n
170	51	170	51)	Separator)_Enum		

In order to merge into a statement like this, the explicitly coded parts and the implicit parts need to be explicitly created in preparation for the merger. This concept of a Blaise statement being expanded so all the parts of the statement exist is being referred to as an expanded Blaise Data Object, or simply, Blaise Data Object (BDO). The BDO is a key concept in the source editor design. With the BDO we can modify the explicitly coded parts along with the implicit parts of a Blaise statement.

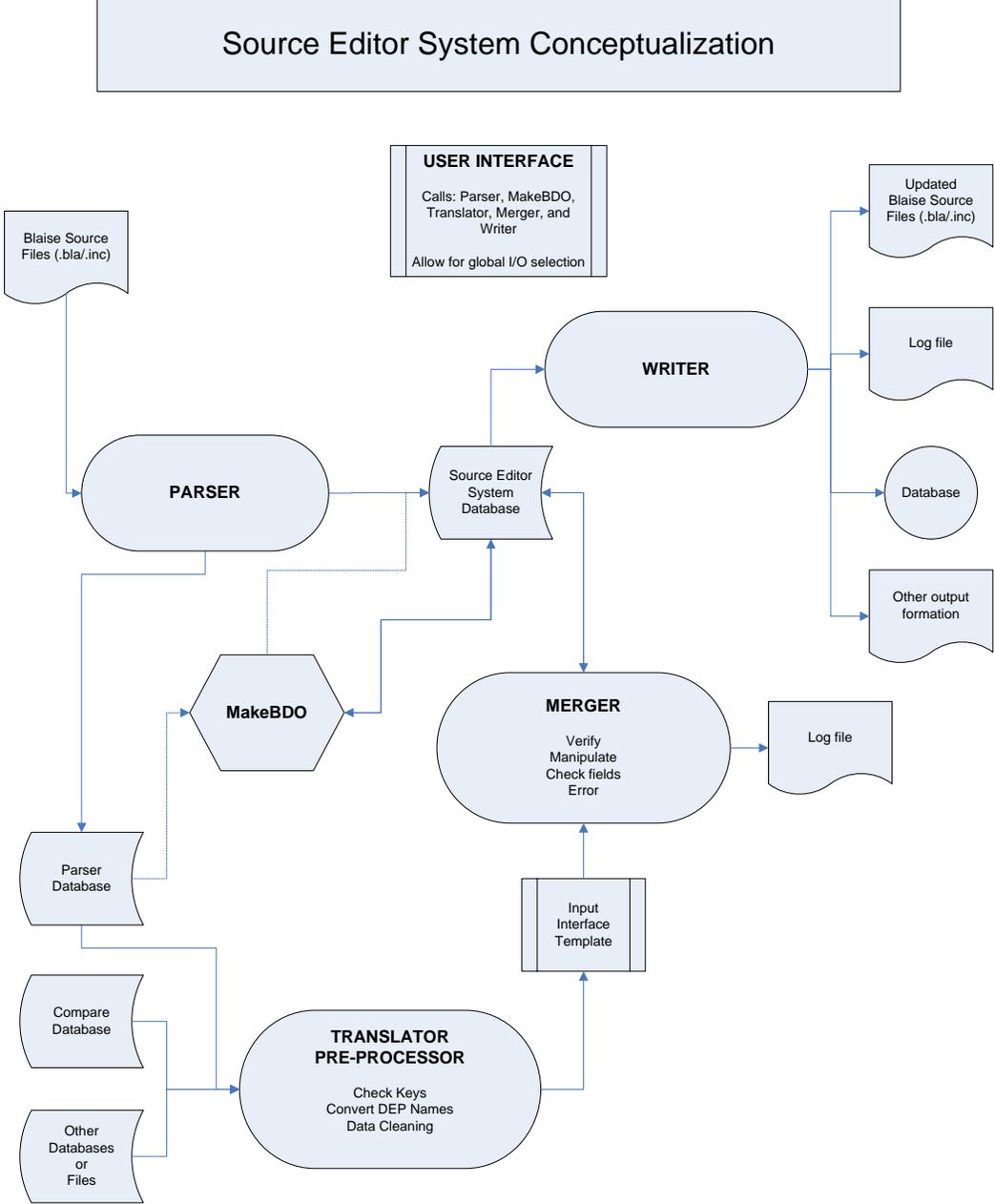
Whitespace and comments in Blaise are statements that stand alone. They're not associated with a field, a block, or a user defined data type. Comments and whitespace are not statements we can locate by using a field name or user defined data type name. They cannot be updated by the merger.

As for deletions, we can delete anything that exists for a Blaise statement, as long as the statement and parts of the statement can be uniquely identified.

As for the assumptions, a few are being made. One assumption is that the user data is in a structured electronic format. Another assumption is that a key, like field name, is provided with the update request. A third assumption is that more than one type of merge key is needed, and that it depends on the token type. The last assumption is that the merger will use an input format that will handle all requests. To see an example of the token types, merge keys, and the user input format, please refer to the appendix.

One question that came up often during the source editor design was “Why don’t we use the .bmi?” The appeal of the .bmi is that Blaise has already done the parsing and tokenization of the statements from the source code files. It is in a form programmers working with the Blaise Application Program Interface (API) are familiar with. It is probably possible to create Blaise source code from the .bmi. However, the resulting source code files would lose the include file structure, comments and whitespace. For HRS’ purposes it is important to retain the original include file structure, comments and whitespace. Due to these needs we cannot use the .bmi.

Figure 1 Source Editor Conceptualization



4. Design Implementation

Our general design implementation objective was to think of the core system components as modules, namely, a modular design approach. We did not want to create unnecessary or redundant functions across components. We wanted to be able to have reusable code, i.e., not so customized that we would have to open the source editor system code to change I/O specifications. More importantly, our design had to be able to handle the HRS CAI.

Our three main implementation objectives were: to create the BDO (Blaise Data Objects), merge the updates, and write out the new files. In order to create the BDO, we had to parse the Blaise source code files to know what is defined in the datamodel. We needed to add all of the potential space for the expanded statements. The merger would then take the user input data and merge it into the expanded statements in the BDO. Last, we would write out the new files with all of the updates.

In the design implementation, we programmed the basic components of the system separately. They are: the parser, MakeBDO⁴, the pre-processor (translator), the merger and the writer. The parser and MakeBDO functions prepare the table for all possible expanded syntax. The pre-processor writes all user updates into one table (user input format) that is used by the merger to update the expanded syntax in the BDO. The writer is the last component in the system.

Taking into account the complexity of the tasks, the general design objective of a modular design, and the division of work tasks, we decided that trying to cram the statement expansion and language reordering into the parsing routines would be awkward, complex, confusing, and in any case not directly related to the parsing process. The language reordering and the statement expansion are not really part of the merging process either. So we decided we had to have an intermediate process of making a BDO. There were clear advantages of using separate components for the different functions in the source editor, including making debugging and application maintenance much easier.

Given the scope of work that needed to be done, the time frame, and the resources available, we could not implement the whole design. We decided to do a phased-in implementation, where the programming effort would focus on core elements of the system. It was important for us to consider the addition of a new language in our application design and implementation. We called this Phase 1.

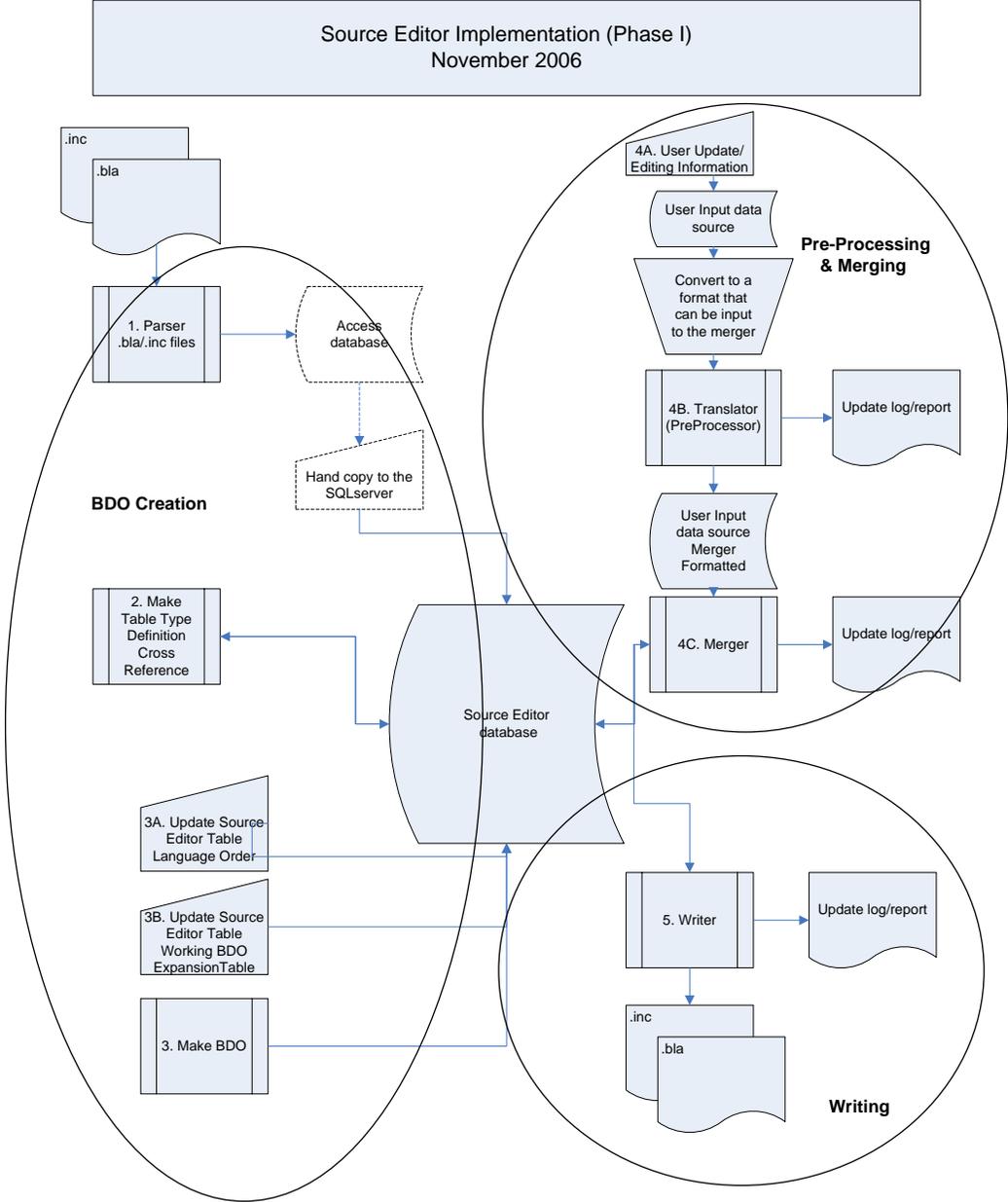
An implementation assumption we made is that the initial Blaise source files should compile. Thus, the parsing routine does not check for Blaise syntax errors. If the Blaise files don't compile after being run through the source editor, the user input data needs to be checked. Routines to check user input data will be taken into consideration when working on Phase 2 of the source editor system. Phase 2 developments will focus more on the translator, pre-processing routines, and the anticipated variability of user input files.

⁴ MakeBDO is the application module that creates the BDO.

We currently have a working implementation of the source editor system's core functions. The next section covers the Phase 1 implementation modules of the source editor system.

5. Implementation Modules

Figure 2 Implementation Design



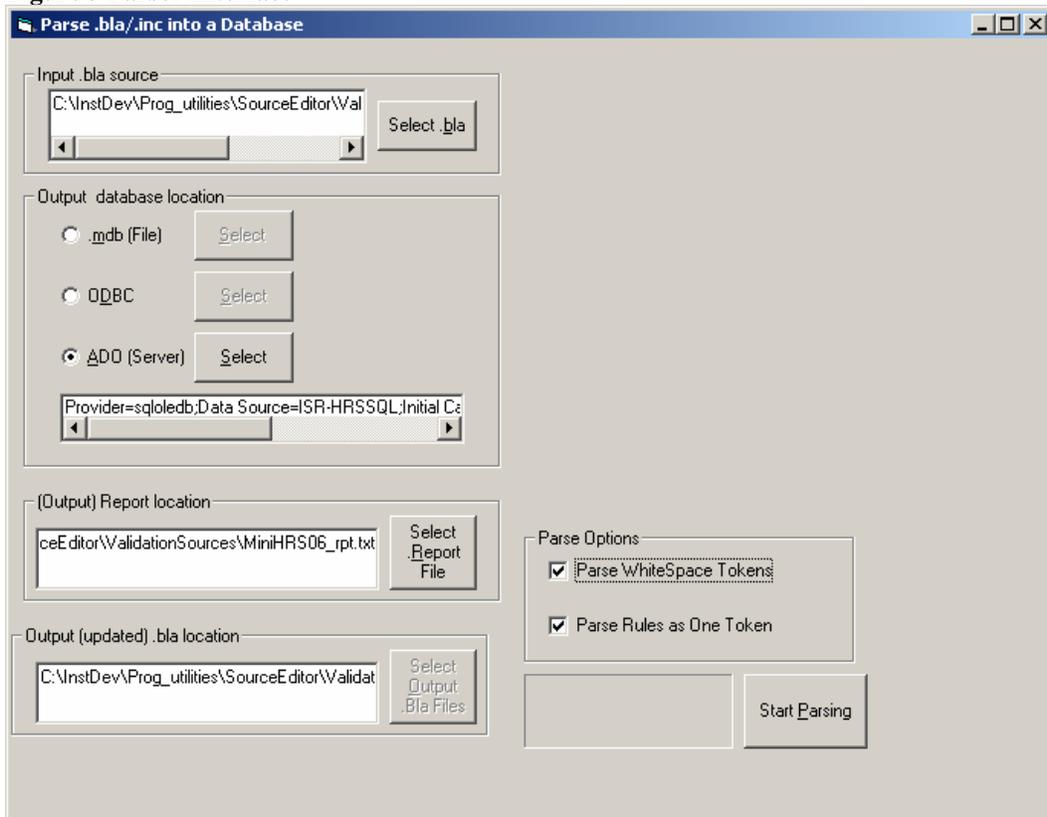
5.1 Parser

The first process in the source editor is parsing the .bla and .inc files into tokens. Tokens are defined parts of Blaise syntax statements. The parser can be used as a stand alone application to provide information about the block structures and logic structures. It has the ability to write to an MS Access database as well as to a SQLServer database. The

parser application is used to create the tables and information that is the base for the BDO table creation.

For the parser to generate output for the source editor system, the 'Parse WhiteSpace Tokens' option must be selected. 'Parse WhiteSpace Tokens' allows for the capture of tabs, spaces, carriage returns, etc. that are essential in order to preserve the whitespace in output files. This feature is optional and can remain unchecked if the parser database is to be used for analysis purposes only. 'Parse Rules as One Token' is also optional. Selecting it helps to reduce processing time and reduces the number of records generated when there are no anticipated updates to the code in the rules.

Figure 3 Parser Interface



The parser application is run -- after selecting an input .bla source, the output database location, and parse options -- using the 'Start Parsing' button. The parser indicates when it is done.

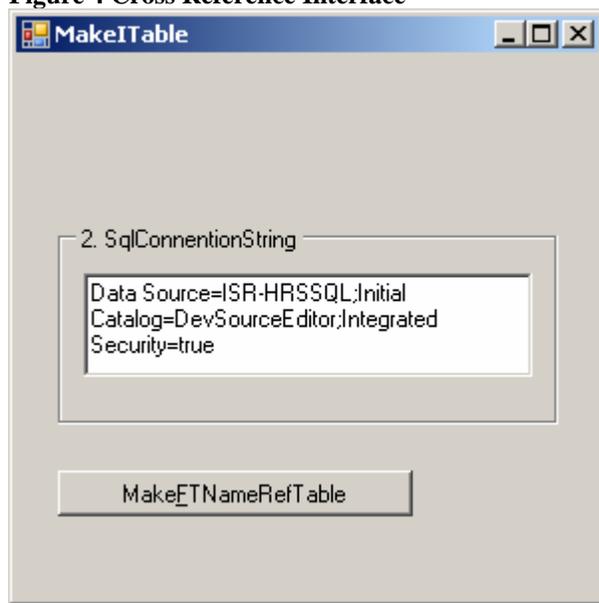


To see examples of tables populated by the parser, refer to the appendix.

5.2 Table Type Definition Cross Reference

In preparation for making the BDO ready for the merger, an application creates references to type definitions that are not included in the same location as the field. Some types are used in many places, but defined in only one place. The interface for the application is shown in Figure 4.

Figure 4 Cross Reference Interface



In the example below for field TP50 the enumeration type is defined at the field. For field TP60 the enumeration type is defined elsewhere. The cross reference table provides this information to the BDO. See Table 3 for an example of data in the cross reference table.

Fields

TP50 "What type of fish do you have?": set[3] of (n "none", f "fresh water", s "salt water")

TP60 "What type of mammals aside from dogs or cats do you have?": TMammals

Table 3 Type Definition Cross Reference

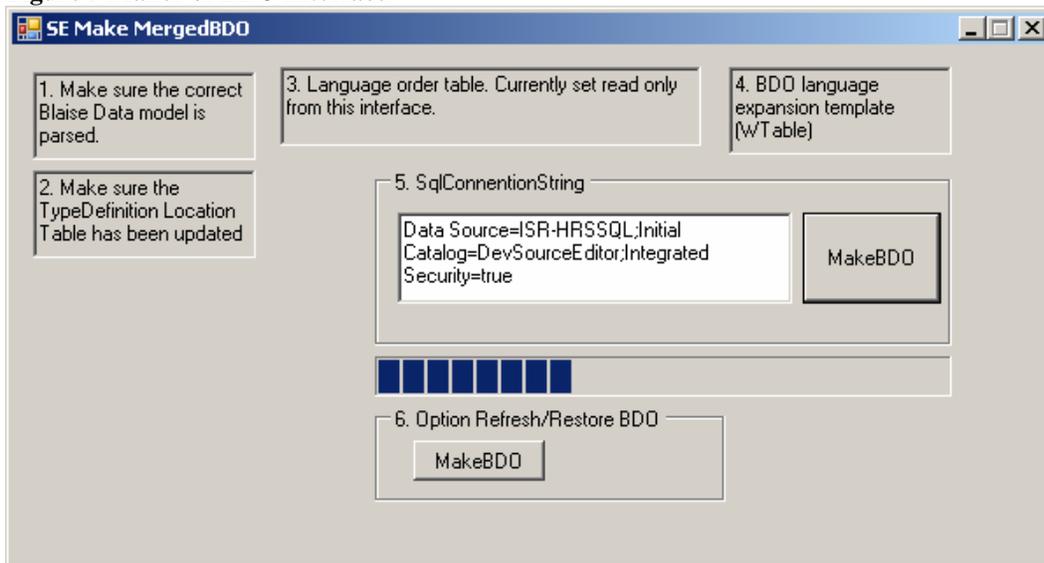
Blkname	Field Name	Type Name	Type No	Blk path	Blk No End	Blk NameEnd	Type NameEnd	TokenType
SE_illustration		tmammals		1..	1	SE_illustration	tmammals	TName
B2_Other	TP50		40					
B2_Other	TP60		41	8.1..	1	SE_illustration	tmammals	FTName

5.3 Creating the BDO (MakeBDO)

The MakeBDO application performs several functions. It creates anchors or spaces for Blaise statement tokens that are not explicit in the parsed code. It allows the user to indicate which implicit tokens to create. It has the capacity to reorder the languages. The MakeBDO application creates a table, which is used by the merger to write updates and by the writer as a data source.

Prior to running the application, in Figure 5, step 1 refers to tables made by the parser; step 2 refers to the cross reference table made by the stand alone Type Definition Cross Reference application; in step 3 the language order table is checked and adjusted if necessary for reordering languages. We knew we would get into volume, scale, and processing time issues, so in step 4, we provided the ability to indicate the number of languages⁵ we want to allow space for in the expanded statement tokens for the BDO. All of these tables are used in the creation of the BDO.

Figure 5 Maker of BDO Interface



“Option refresh/restore BDO” is used to re-establish the BDO table before a merge. This shorter process is used for instances where a bad merge⁶ was made. This is especially useful on a large datamodel because it saves time.

5.4 Translator / Pre-processor

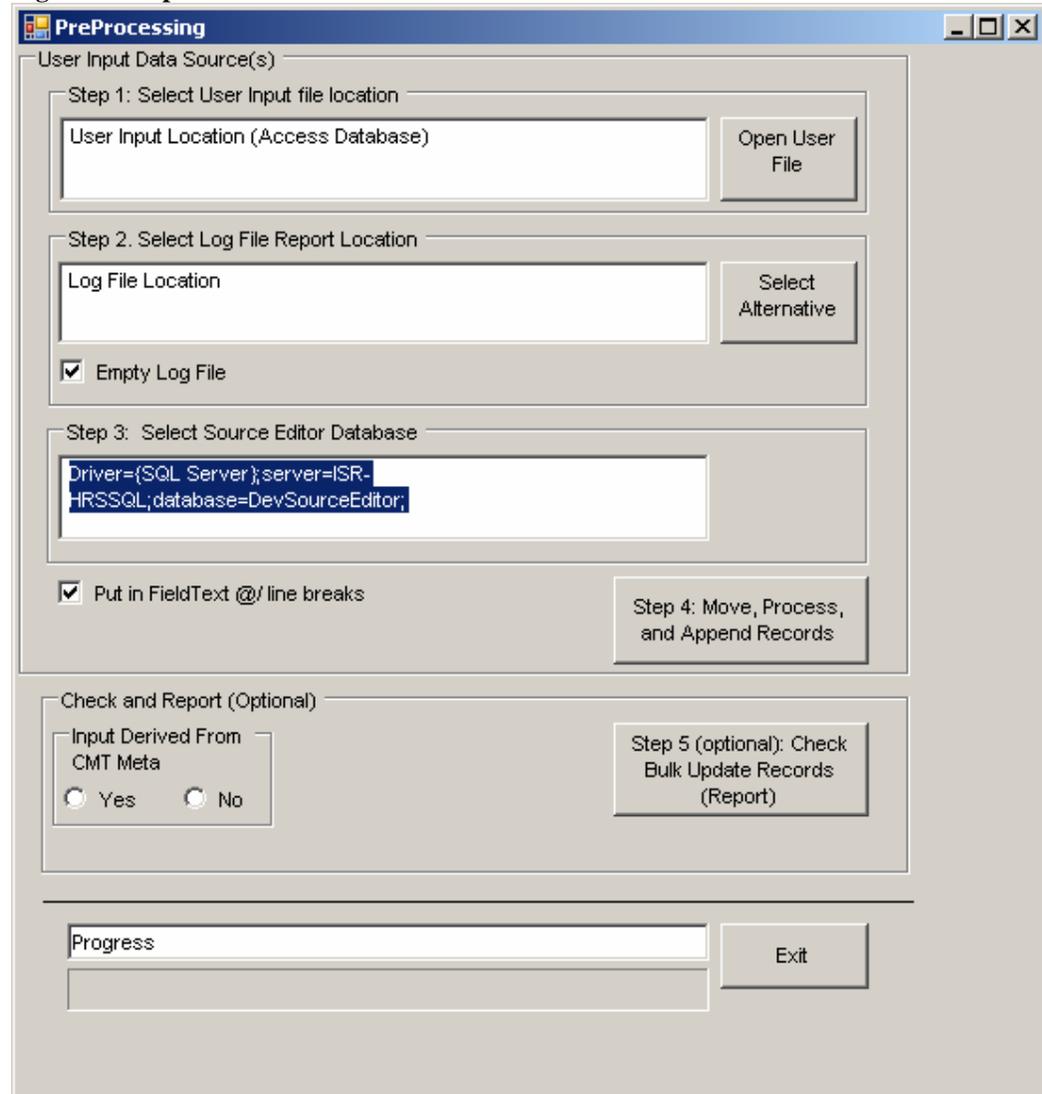
The goal of the translator or pre-processor is to move data into a table that the merger uses to make data updates into the BDO table. In Phase 1, HRS used a customized pre-processor without a translator. A translator is needed to convert long DEP fieldnames into defined blocks and fieldnames. A translator is a major component planned for future development.

⁵ HRS has seven languages.

⁶ A bad merge would occur when the user update information is not specified correctly via bad data or bad format.

How did we get away without writing a translator in Phase 1? Our updates mainly consisted of adding a new language. For this task the user input data was a very clean data source with the Blaise block path and field names, so we didn't have to translate DEP fieldnames.

Figure 6 Pre-processor Interface



The option 'Put in FieldText @/ line breaks' is used to modify user input for question text. It converts '@/' to <linebreak>+ '@/' to have the line break in the typical manner for HRS.

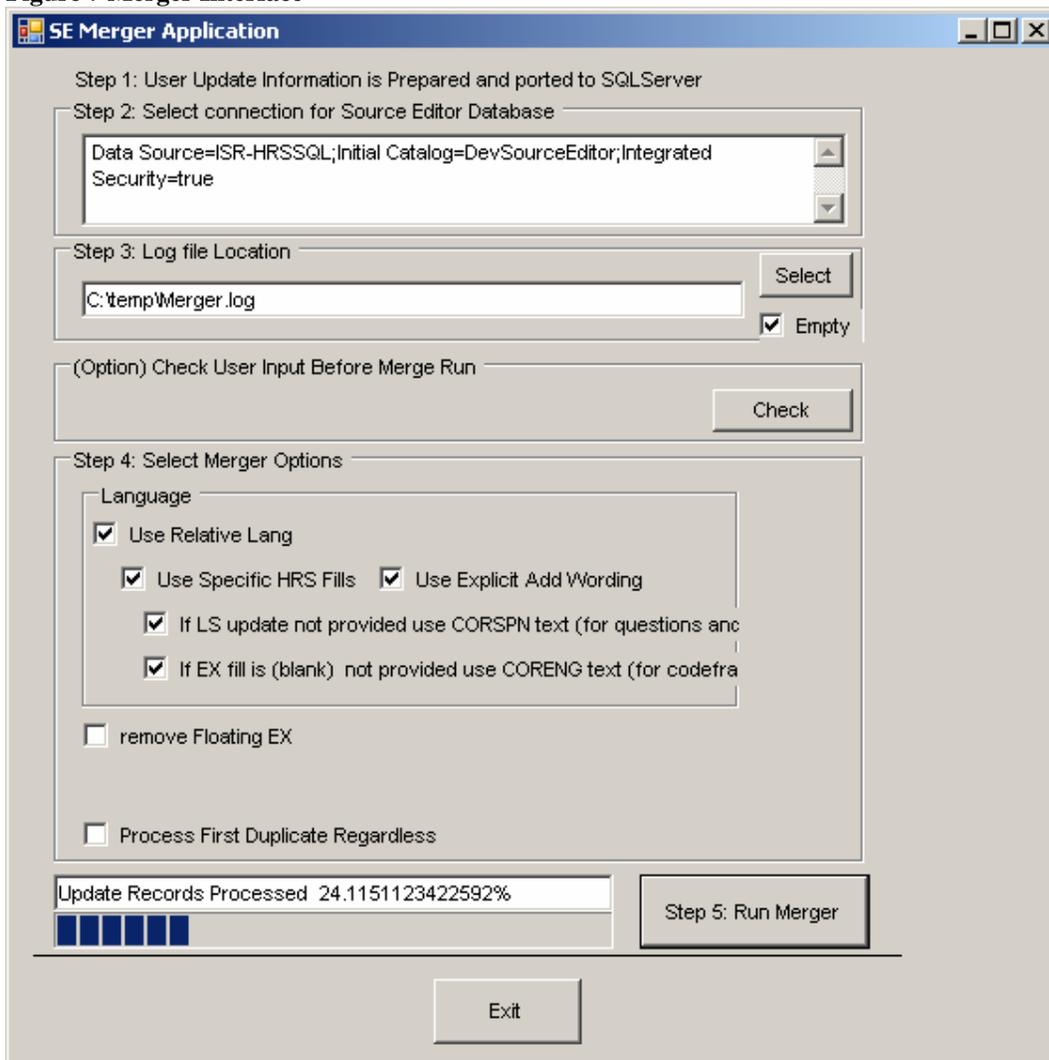
Some of the pre-processes to prepare user input data for merging are:

- placing quotes around text
- explicitly stating Language ID (LID)
- checking for duplicate requests
- checking for items referenced multiple times but only appearing once in code
- checking for duplicate block names
- checking for token type

5.5 The Merger

The merger application processes changes or requests and merges them into the BDO table.

Figure 7 Merger Interface



'Use Relative Language' and 'Use Explicit Add Wording' are generic options that can be used with any instrument. 'Use Relative Language' is selected when LID is not used in code. 'Use Explicit Add Wording' writes automated quoted text wherever it is needed in the expanded statements. All other options are HRS-specific.

The option "Check User Input Before Merge Run" is optional. This option writes a report with checks on language order; expansion statements, duplicate blocks and duplicate include files. To see an example of the report produced by this option, refer to the appendix.

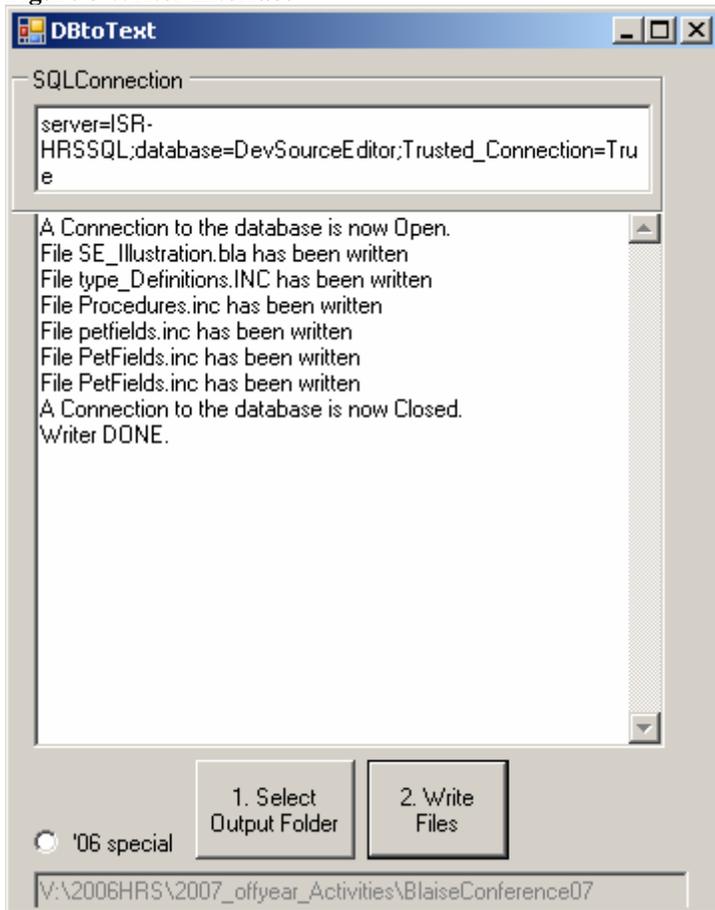
The option 'Process First Duplicate Regardless' allows the processing of duplicate user update requests.

The merger writes a log file with information about run time, number of token type records, and user input updates that were not processed. To see an example of the log produced by this process, refer to the appendix.

5.6 The Writer

The writer takes records from the BDO table and creates updated .bla and .inc files. If no updates and no language re-ordering are applied, the .bla and .inc files will write out unchanged⁷.

Figure 8 Writer Interface



The writer produces a log indicating the name of each file produced and the corresponding number of updates, if any. This is a sample log:

File SE_Illustration.bla has been written with 8 updates.
File type_Definitions.INC has been written with 57 updates.
File Procedures.inc has been written with 0 updates.
File petfields.inc has been written with 18 updates.
File PetFields.inc has been written with 18 updates.
File PetFields.inc has been written with 18 updates.⁸

⁷ As expected, initial source files run through the source editor without user updates will output with no changes.

6. System Interface

Future plans include creating a system interface that encompasses all system functions through one interface organized by tabs that will reference each component's interface, and include instructions on how to perform critical checks.

7. Testing

Good application development has a testing plan to ensure a quality product. We planned for three types of tests: the first test was to test each component or program module separately; each programmer did rudimentary component testing prior to using the validation datamodel. The second test was to integrate the components to test for interoperability and general system functioning. The final test dealt with size, scalability and throughput issues.

As input to the testing process, we created validation files containing Blaise source code designed to model all the functionality that our programs were supposed to have. Validation files are preferable to user inputs that either do not exercise all the functions of the application or produce multiple errors for functions that are used. Our last step was to run large volume test. Since these tests take time and resources, they should not be done until after the basic functionality testing for each piece is complete.

The validation source files for the source editor system are too large to include in this paper. We have small representative Blaise Program Source Code files covering key concepts in the appendix of this paper.

8. Is It Worth It? Current Use and Future Plans

Is it worth it? The main reason we created a successful application is because it meets the goal of doing large scale updates faster and more accurately than doing them by hand. Our rough estimates of the time differences between doing tasks by hand and doing them using an automated method such as the source editor indicate that it's worth it.

The example of changing descriptors by hand involves the steps of typing in the field name, searching the code for the field, copying the new descriptor from the user update file, deleting the existing descriptor, and pasting the descriptor into the source code. Doing these highly repetitive steps 2,700 times increases the likelihood of repetitive use injuries (a non-trivial issue), has more potential for errors due to typing and searching techniques used, and would take about 165 hours.

The system is even more useful in dealing with large scale changes. As noted above, during the most recent development effort a new language was added to the instrument. The addition of a new language affects all fields and all enumerated types. This is a much larger task than changing descriptors. The HRS CAI application is a very large application. It has 175,600 lines of code covering 56 files and 5,600 uniquely defined fields (excluding auxfields, locals, and parameter fields). To parse the whole application,

⁸ Since the file was included three times, it is written three times by the writer. This can be problematic if different changes are requested for blocks and fields that map into the same include file.

make 6,500 updates, and write the updated files took about 24 hours using the source editor.

Table 4 Source Editor Scale and Time Estimate

Scale of Task	Approx time for task
Parsing 175,600 lines of code (56 files)	8 hours
Type Cross Reference	2 hours
Create BDO	4 to 6 hours
Merge (approx 6500 updates)	6 hours
Write files	3 minutes
Total Time	24 hours

The time difference is a few days versus a few weeks. Since most of the work is being done by machine, updates are more accurate, the likelihood of repetitive use injuries is reduced, and time is saved, by not having to replicate information that is already in a structured electronic format. If large scale tasks are done by hand a few times every other year (as in HRS), the time invested in developing a source editing system is justified.

9. Conclusion

The source editor system design conceptualization, development, and implementation encompassed design features such as a parser, a maker of expanded Blaise statements, a merger, a user data translator/pre-processor and a writer. The elements work together to make a system that edits Blaise source code files, processing updates on a very large scale.

Although we plan to add functionality to the system in the next development phase, the source editor can handle all of the following tasks without further development of the core functions:

- Strip out obsolete or dated comments from prior years.
- Update tags. Modify tags to be more descriptive.
- Update descriptors. Modify labels for data out.
- Update data types. Modify field size, field ranges, etc.
- Update language text. Text provided by another system such as a product from the HRS translation group.

As currently implemented, the Source Editor System benefits are:

- Time saving, resulting in faster turn-around of tasks.
- Hundreds of changes can be made at one time.
- More accurate placement of updates and therefore better quality.
- May reduce repetitive-use injury.
- Robust enough to handle applications as large as HRS.
- Generic enough to handle other non-HRS Blaise CAI applications.

- The application can add or re-order languages.
- The application has features to help handle scale issues.

Along with all these benefits, the Source Editor updates the .bla and .inc files automatically, while preserving comments, whitespace, and the .bla and .inc file structure.

10. Appendix

This appendix is not for the casual reader. The main document gives the overview of the design concepts. The material in the appendix gives a cryptic view of implementation design details that most readers would find, uninteresting, too detailed, or undocumented to make sense of. This information would make sense to someone that can look at tables and ascertain primary and secondary key relations determined by various combinations of columns.

10.1 Sample Blaise Program Source Code Files

Four files make up the sample illustrative program. These files will be used to show how tables are populated. The tables will be included. Records in the tables may be excluded to save space.

These files are used to illustrate things as:

- How the parser reads in lines and files
- Duplicate block names
- Types defined at the field versus types defined elsewhere
- Relative language order for text
- Explicit language order by using LID
- HRS conventions of using special comments and to denote language by relative position.

10.1.1 FileName: SE_Illustration.bla

```
DATAMODEL SE_illustration "Illustration of Source Editor Ideas"
```

```
LANGUAGES =  
  Eng "English",  
  SPN "Spanish",  
  AltEng "Alternative English",  
  AltSpn "alternative spanish" {to be used in the future}
```

```
ATTRIBUTES= nodk, norf, noempty
```

```
INCLUDE "type_Definitions.INC"
```

```
INCLUDE "Procedures.inc" {See what happens when a program line with include has  
another statement in it.}
```

```
TYPE tmammals=(n "none", {This is where the that needs to be looked up is found --  
see TP60}  
  h "hamster",  
  f "ferret",  
  o "other")
```

BLOCK B1_pet "Type of pet"

INCLUDE "PetFields.inc" {Note: This file is included several times under different blocks to illustrate the 'duplicate' file concept}

RULES

```
tp1
IF tp1=S or tp1=y or tp1=dk then
  tp2
endif
tp3
if tp3<>no or tp3=dk then
  tp5
endif
```

ENDBLOCK

BLOCK B2_Other "OtherType of pet"

PARAMETERS IMPORT pipt:string

INCLUDE "PetFields.inc" {Note: This file is included several times under different blocks to illustrate the 'duplicate' file concept}

FIELDS

TP40 (T4) {This is an example of relative language positioning.}

{eng} "This is a special question about your other ^pipt:

@/@/Why do you have this pet? "

{spn} "Esta es una pregunta especial sobre sus otros ^pipt:

@/@/Why do you have this pet? "

{alteng} "This is a special question about your other ^pipt:

@/@/Why do you and your family have this pet? "

/"Why this pet":

open

{mentioned in the paper to show an enumeration defined at a field and
and enumeration defined elsewhere that needs to be looked up}

TP50 "What type of fish do you have?": set[3] of (n "none", f "fresh water", s "salt water"
)

TP60 "What type of mammals aside from dogs or cats do you have?": TMammals

ENDBLOCK

Fields

Q1 (Q1) "Are you ready to answer questions?":(y,n) {mentioned in the paper to
show a statement}

Q2 (Q2) "Do you have any pets?" :TYesNoV2

Q3 eng "How many pets do you have?" /"Number of pets"
:SERange1, dk

```
DogPets:B1_pet
CatPets:B1_pet
RodentPets:B1_pet
```

```
otherPets:B2_Other
Q4 (E1) {eng}"Thank you" {spn} " " {alteng} "THANK YOU"
:(continue)
```

```
LOCALS pettype, p:string
```

```
RULES
```

```
Q1
```

```
IF q1=y or q1=dk then
```

```
Q2
```

```
    if q2=y or q1=dk then
```

```
        q3
```

```
        p:='dogs'
```

```
        TXT_Petype(p,pettype ) {HRS uses procedures to fill text versus putting it in
the rules}
```

```
        DogPets
```

```
        pettype:='cats'
```

```
        catPets
```

```
        pettype:='rats'
```

```
        RodentPets
```

```
        pettype:='not mentioned'
```

```
        otherPets ( pettype)
```

```
    endif
```

```
ENDIF
```

```
ENDMODEL
```

10.1.2 FileName: type_Definitions.INC

```
type
```

```
{V1 has 'gaps' in the statement elements which need to be filled in by the BDO}
```

```
  TYesNoV1 =( Y (2) "yes",
```

```
              N (3) "no" )
```

```
{V2 has all statement element completely filled in}
```

```
  TYesNoV2 = (Y (1) eng "yes" spn "si" alteng "YES",
```

```
              N (5) eng "no" spn "no" alteng "NO"), dk, norf, empty
```

```
  SERange1= 1..15, rf
```

```
Tmonth =( jan (1) "january",
```

```
          feb (2) "february",
```

```
          mar (3) "march",
```

```
          apr (4) "april",
```

```
          may (5) "may",
```

```
          jun "june",
```

```
          jul "july",
```

```
          aug "august",
```

```
          sep "september",
```

```
oct "october",
nov "november",
dec "december")
```

```
Tinteger = -999..999
T2005ToPresent =(t2005,
t2006, t2007)
```

```
{just to show some of the Spanish diacriticals}
Tspnspc =( s "Sección",
e "él" ,
u "en los últimos dos años"), dk
```

10.1.3 FileName: Procedures.inc

```
PROCEDURE TXT_Petype
parameters
inword:string
export outword:string
{translates the fill text between english and spanish}
RULES
```

```
IF inword='dogs' then
  if activelanguage <> eng and activelanguage <> alteng then
    outword:='perros'
  ELSE
    outword:=inword
  endif
ELSEIF INWORD='cats'
  then outword:='gatos'
else
  outword:=inword
ENDIF
endprocedure
```

```
Block BA3
{This block is not used. It's just to show an include file being called by an include file}
include "petfields.inc"
endblock
```

```
BLOCK BB "BB version 2" {an example of a duplicate block name}
fields bb1 "Pick a number between 1 and 3 ":1..3, empty, dk, rf
ENDBLOCK
```

```
Table BC
fields
  BC "put some words in . . .":open
  Block BB "BB version 1" {an example of a duplicate block name}
  fields bb1 "Pick a number between 1 and 5 ":1..5, empty

endblock
```

ENDTABLE {Blaise is not strict on the end token for Blocks, Procedures, Datamodel, and Table}

10.2 Overview of Tables and Procedures used by the Source Editor System

This table is a list of procedures and tables used by Source Editor System programs, with a brief description of the table or procedure and a column listing which application module uses it. The column on the far left indicates whether a table or procedures appears in the appendix.

In-cluded in the appendix	Tbl or Proc	Name	Brief description of table or procedure	P A R S E R	L O O K - U P	M A K E B O D O	Pre - P R O C E S S	M E R G E R	W R I T E R
y	T	Blocks	Blocks, Prodecures, Tables, and DataModel defined	x	x	x			
y	T	FieldType	Fields, Auxfields, Parameter Fields, Locals and the parsed type number associated with the field	x		x			
y	T	FileLines	Lines read from the .bla and .inc files while parsing through the data model	x		x			
y	T	Files	the file(s) that started the parsing and all included files	x		x			
n	T	LgcBlocks	the logic blocking structure	x					
y	T	Token	The tokenized Blaise Statements	x	x	x			
n	T	TypeCode	Enumerated code name and implicit/explicit code number	x					
n	T	TypeLEQD_language	All language text for fields, descriptors and enumerations	x					
n	T	TypeNCDKRFE	The dk/nodk, rf/norf, empty/noempty associated with a type	x					
n	T	TypeOf	Parsed type number and description of user defined or system defined datatype	x		x			
n	T	TypeSet	Parsed type number and description of type defined as an array or set	x					
n	P	SEB00	empties table ITTemplate		x				
n	T	ITTemplate	Table structure template use to write data to		x				
y	P	SEB01	Puts together information from tables and populating the table with information needed to do the look-up		x				
n	P	SEBFT00	empties table FT		x				
n	T	FT			x				

y	P	SEBFT01	Selects the records of interest need for the look-up process and puts them in FT		x				
y	T	ftRefTbl	Resultant look-up table after program does the work using the FT table		x	x			
y	T	Wtable	Work table specifying how to expand/make BDO space for languages and language identifiers			x			
y	T	BDOVariantsUniverse	Contains the blaise statement token order			x			
y	T	LanguageOrder	Contains the language order in the parsed datamodel and the language order to be used when 'writing' /creating the MergedBDO table			x			
y	T	SE_DatabaseDictionary	Contains the Tokentype definitions and descriptions used in the MergedBDO Table			x			
y	P	MakeTokeBDOPrep	Puts together information from tables and makes a view used by the program which makes the mergedBDO			x			
n	T	BDOVariantsUniverseRoll	A rolled-up, collapse view of information from the table BDOVariantsUniverse			x			
n	T	dt40	a temp table used by the program making the MergedBDO to reorder language			x			
n	T	TokenDBOTest	a temp/test debug/back up copy of BDO records prior to having columns renamed and tokentypes created to match the SE_DatabaseDictionary tokentypes			x			
n	T	tta	a temp table			x			
y	P	MakeMergedBDO	puts information together from tables to make the mergedBDO table			x			
n	T	DupFilesB	Table with Duplicate Block Names			x			
n	T	DupFilesA10	Table with Duplicate Include file use			x			
n	P	SE0_makeFromTest	Puts tables together to make table TTA			x			
n	P	SE5_1_S0	empties column TokenTypeBDO in TTA			x			
n	P	SE5_1_S1	Puts in the SE Dictionary token type name			x			
n	P	SE5_2_S2	Puts in the SE Dictionary token type name using another criteria			x			
n	P	SE5_3_S	Puts in the SE Dictionary token type name using another criteria			x			
n	P	SE5_4_S	Puts in the SE Dictionary token type name using another criteria			x			
n	P	SE6_1_F	Puts in the SE Dictionary token type name using another criteria			x			
n	P	SE6_2_FA	Puts in the SE Dictionary token type name using another criteria			x			
n	P	SE6_3_FL	Puts in the SE Dictionary token type name using another criteria			x			
n	P	SE6_4_FP	Puts in the SE Dictionary token type name using another criteria			x			
n	P	SE6_5_t	Puts in the SE Dictionary token type name			x			

			using another criteria						
n	P	SE7	Puts in the SE Dictionary token type name using another criteria			x			
n	P	SE8_1_clean	Cleaning up data in a column left over from the processing program			x			
n	P	SE91	Puts in the SE Dictionary token type name using another criteria			x			
n	P	SE92	Puts in the SE Dictionary token type name using another criteria			x			
n	P	SETypeNoEndZero	Cleaning up data in a column left over from the processing program			x			
n	P	SEMergerBDOKey	the table needs to have keys defined			x			
n	P	MakeftRefTblRollup	The procedure to make the table			x			
n	T	ftRefTblRollup	A rolled-up, collapse view of information from the table ftRefTbl			x			
y	T	MergedBDO	The table to merge updates into and to write revised files from			x		x	x
n	T	AllCodeMergInfo	Special Input for LS language update				x		
n	T	HRS06_cod_LANGUAGES	Special Input for LS language update				x		
y	T	UserInputFormat	User/Bulk updates to be source files				x	x	
n	T	UserInputProcessed	UserInputFormat table with processing updates to match the merge log of records not processed						tbd
n	T	UserInputFormat_Enum	preprocessor input for PS generated var file					tbd	
n	T	UserInputFormat_Qtext	preprocessor input for PS generated cod file					tbd	

10.3 Parser Tables⁹

10.3.1 Table: Files

This table keeps track of the files used or included in the datamodel. Each time a file is referenced or included a unique file reference number, FLRefNo, is generated. An include file used several times will have more than one file reference number associated with a file name, see File Name ‘petfields.inc’.

This table also has information as to which file called the include file, FINestPath¹⁰; how deep or nested the call is, FINest; and the number of lines in the file, FILnCnt.

⁹ The input sample source files were modified for better readability after the tables were generated. The data in the tables will differ slightly from source files.

¹⁰ The path read left to right moving from the lowest level to the highest. For example file reference number 4, was include from file reference number 3, Procedures.inc, which was included from file reference number 1, SE_Illustration.bla.

Aside: If you want to rename the source editor update files, change the file name in this table prior to making the MergedBDO or change the filename in the MergedBDO table.

FIRefNo	FIName	FIPath	FIDate	FILnCnt	FINest	FINestPath	FLBegTime	FLEndTime	FLTimeElapsed
2	type_Definitions.INC	V:\2006HRS\2007_offyear_Activities\BlaiseConference07\SourceEditorPaper\BlaiseExample\	07/23/2007 7:49:56 PM	34	2	2.1.	589977546	589983218	5672
4	petfields.inc	V:\2006HRS\2007_offyear_Activities\BlaiseConference07\SourceEditorPaper\BlaiseExample\	07/23/2007 3:45:30 PM	12	3	4.3.1.	589984281	589986468	2187
3	Procedures.inc	V:\2006HRS\2007_offyear_Activities\BlaiseConference07\SourceEditorPaper\BlaiseExample\	07/24/2007 3:47:18 PM	41	2	3.1.	589984281	589988125	3844
5	PetFields.inc	V:\2006HRS\2007_offyear_Activities\BlaiseConference07\SourceEditorPaper\BlaiseExample\	07/23/2007 3:45:30 PM	12	2	5.1.	589989093	589991015	1922
6	PetFields.inc	V:\2006HRS\2007_offyear_Activities\BlaiseConference07\SourceEditorPaper\BlaiseExample\	07/23/2007 3:45:30 PM	12	2	6.1.	589991718	589993640	1922
1	SE_Illustration.bla	V:\2006HRS\2007_offyear_Activities\BlaiseConference07\SourceEditorPaper\BlaiseExample\	07/26/2007 2:59:08 PM	99	1	1.	589991718	589997812	6094

10.3.2 Table: FileLines

This table contains the file lines as read by the Blaise program. The line number, LnNo, is a unique program line identifier. This is necessary because an include file can be referenced more than once. The line content, of an included file, can have a different meaning depending on how it's included. (see petfields and field TP1 as an example. In one case field TP1 is defined under block B2_Other. In another case field TP1 is defined under block BA3.

Note: The 34+12+26+93 lines of code from the source files translate into 34 + (12x3) + 26 + 93 lines of code due to multiple including of the file petfields.inc

Note: The file reference number, the line number within the include file does not necessarily follow the sequence of the program line number, LnNo.

Note: Some records have been removed from the table records for brevity.

FIRefNo	LnNo	LnLength	FILnCnt	FILn
1	1	63	1	datamodel SE_illustration "Illustration of Source Editor Ideas"
1	2	0	2	
1	3	25	3	languages =Eng "English",
1	4	16	4	SPN "Spanish",
1	5	34	5	AltEng "Alternative English",

FIRefNo	LnNo	LnLength	FILnCnt	FILn
1	6	0	6	
1	7	61	7	altspn "alternative spaniah" {to be uswd in the future}
1	8	0	8	
1	9	31	9	{Add AltSpn language later}
1	10	31	10	attributes= nodk, norf, noempty
1	11	30	11	include "type_Definitions.INC"
2	12	0	1	
2	13	4	2	type
2	14	87	3	{V1 has 'gaps' in the statement elements which need to be filled in by the BDO}
2	15	25	4	TYesNoV1 =(Y (2) "yes",
2	16	25	5	N (3) "no")
2	17	0	6	
2	18	51	7	{V2 has all statement element completly filled in}
2	19	51	8	TYesNoV2 = (Y (1) eng "yes" spn "si" alteng "YES",
2	20	66	9	N (5) eng "no" spn "no" alteng "NO"), dk, norf, empty
2	41	47	30	{just to show some of the spanish diacriticals}
2	42	23	31	Tspnspc =(s "Sección",
2	43	19	32	e "él" ,
2	44	43	33	u "en los últimos dos años"), dk
2	45	11	34	
1	46	0	12	
1	47	44	13	include "Procedures.inc" {it's an odd place}
3	48	21	1	procedure TXT_Pettytype
3	49	10	2	parameters
3	50	13	3	inword:string
3	67	60	20	endprocedure {blaise complier isn't too pick about its ends}
3	68	0	21	
3	69	0	22	
3	70	9	23	Block BA3
3	71	85	24	{This block isn't used. It's just to show an include being called by an include file}
3	72	23	25	include "petfields.inc"
4	73	26	1	{field for the pet blocks}
4	74	32	2	type TYesNoV2 = (Y (11) "yes",
4	75	21	3	N "no",
4	76	29	4	s "Several"), dk
4	77	0	5	
4	78	50	6	fields TP1 eng "For this type of pet, ^pettype, @/
4	79	62	7	@/Do you have any? " alteng"For this type of pet,

FIRefNo	LnNo	LnLength	FILnCnt	FILn
				^pettype, @/
4	80	30	8	@/Do you have any? ":TYesNoV2
4	81	44	9	TP2 "How many ^pettype do you have?": 0..100
4	82	74	10	tp3 (Pet3) "Is there anything special you want to mention about ^pettype?"
4	83	27	11	: (Yes,Maybe, No) , dk, rf
4	84	45	12	TP5 (PT4) "If so what you like to say?": open
3	85	8	26	endblock
3	86	0	27	
3	87	62	28	Block BB "BB version 2" {an example of a duplicate block name}
3	88	64	29	fields bb1 "Pick a number between 1 and 3 ":1..3, empty, dk, rf
3	89	8	30	endblock
3	90	0	31	
3	91	8	32	table BC
3	92	0	33	
3	93	6	34	fields
3	94	34	35	BC "put some words in . . .":open
3	95	1	36	
3	96	63	37	Block BB "BB version 1" {an example of a duplicate block name}
3	97	56	38	fields bb1 "Pick a number between 1 and 5 ":1..5, empty
3	98	0	39	
3	99	9	40	endblock
3	100	8	41	endtable
1	101	44	13	include "Procedures.inc" {it's an odd place}
1	102	0	14	
1	103	24	15	type tmammals=(n "none",
1	104	26	16	h "hamster",
1	105	25	17	f "ferret",
1	106	24	18	o "other")
1	107	0	19	
1	108	26	20	block B1_pet "Type of pet"
1	109	0	21	
1	110	23	22	include "PetFields.inc"
5	111	26	1	{field for the pet blocks}
5	112	32	2	type TYesNoV2 = (Y (11) "yes",
5	113	21	3	N "no",
5	114	29	4	s "Several"), dk
5	115	0	5	
5	116	50	6	fields TP1 eng "For this type of pet, ^pettype, @/

FIRefNo	LnNo	LnLength	FILnCnt	FILn
5	117	62	7	@/Do you have any? " alteng"For this type of pet, ^pettype, @/
5	118	30	8	@/Do you have any? ":TYesNoV2
5	119	44	9	TP2 "How many ^pettype do you have?": 0..100
5	120	74	10	tp3 (Pet3) "Is there anything special you want to mention about ^pettype?"
5	121	27	11	: (Yes,Maybe, No) , dk, rf
5	122	45	12	TP5 (PT4) "If so what you like to say?": open
1	123	6	23	rules
1	124	3	24	tp1
1	125	38	25	IF tp1=S or tp1=y or tp1=dk then
1	135	33	35	block B2_Other "OtherType of pet"
1	136	32	36	parameters import pip: string
1	137	0	37	
1	138	23	38	include "PetFields.inc"
6	139	26	1	{field for the pet blocks}
6	140	32	2	type TYesNoV2 = (Y (11) "yes",
6	141	21	3	N "no",
6	150	45	12	TP5 (PT4) "If so what you like to say?": open
1	151	0	39	
1	152	6	40	fields
1	153	9	41	TP40 (T4)
1	154	61	42	{eng} "This is a special question about your other ^pip:
1	155	43	43	@/@/Why do you have this pet? "
1	156	60	44	Esta es una pregunta especial sobre sus otros ^pip:
1	157	42	45	@/@/Why do you have this pet? "
1	158	64	46	{alteng} "This is a special question about your other ^pip:
1	159	58	47	@/@/Why do you and your family have this pet? "
1	160	0	48	
1	161	16	49	/"Why this pet":
1	162	4	50	open
1	163	0	51	
1	164	93	52	TP50 "What type of fish do you have?": set[3] of (n "none", f "fresh water", s "salt water")
1	165	74	53	TP60 "What type of mammals aside from dogs or cats do you have?": TMammals
1	169	6	57	Fields
1	170	51	58	Q1 (Q1) "Are you ready to answer questions?":(y,n)

FIRefNo	LnNo	LnLength	FILnCnt	FILn
1	210	8	98	endmodel
1	211	0	99	

10.3.3 Table: Blocks

This table contains one record for each block, table, datamodel, or procedure defined. A unique number identifies each block structure.

Note: There are instances where the same block name is defined under a different path. In cases like this the block number must be used to distinguish the block uniquely. See Block name “BB”.

FIRefNo	BlkNo	Blktype	Blkname	Blkdescriptor	Blkpath	Blknest
1	1	D	SE_illustration	"Illustration of Source Editor Ideas"	1..	1
3	2	P	TXT_Pettytype		2.1..	2
3	3	B	BA3		3.1..	2
3	4	B	BB	"BB version 2"	4.1..	2
3	5	T	BC		5.1..	2
3	6	B	BB	"BB version 1"	6.5.1..	3
1	7	B	B1_pet	"Type of pet"	7.1..	2
1	8	B	B2_Other	"OtherType of pet"	8.1..	2

10.3.4 Table: FieldType

For each field, auxfield, local field, or parameter field, the field name is listed, and given a unique field number, FieldNo. Each list of field(s) is assigned a unique type. Most lists of fields have only one element. See how this statement “locals pettytype, p:string” appears in the table below for the fields “pettype” and “p”.

FIRefNo	BlkNo	FieldNo	FieldType	FieldName	TypeNo	FIRefNo	BlkNo	FieldNo	FieldType	FieldName	TypeNo
3	3	0	B	BA3	13	3	4	7	F	bb1	20
3	4	0	B	BB	19	3	5	8	F	BC	22
3	6	0	B	BB	23	3	6	9	F	bb1	24
1	7	0	B	B1_pet	26	5	7	10	F	TP1	28
1	8	0	B	B2_Other	32	5	7	11	F	TP2	29
1	1	0	D	SE_illustration	1	5	7	12	F	tp3	30
3	2	0	P	TXT_Pettytype	10	1	7	13	F	TP5	31
3	5	0	T	BC	21	1	8	14	PI	pipt	33
3	2	1	PI	inword	11	6	8	15	F	TP1	35
3	2	2	PE	outword	12	6	8	16	F	TP2	36
4	3	3	F	TP1	15	6	8	17	F	tp3	37
4	3	4	F	TP2	16	1	8	18	F	TP5	38
4	3	5	F	tp3	17	1	8	19	F	TP40	39
3	3	6	F	TP5	18	1	8	20	F	TP50	40

FIRefNo	BlkNo	FieldNo	FieldType	FieldName	TypeNo	FIRefNo	BlkNo	FieldNo	FieldType	FieldName	TypeNo
1	8	21	F	TP60	41	1	1	27	F	RodentPets	47
1	1	22	F	Q1	42	1	1	28	F	otherPets	48
1	1	23	F	Q2	43	1	1	29	F	Q4	49
1	1	24	F	Q3	44	1	1	30	L	pettype	50
1	1	25	F	DogPets	45	1	1	31	L	p	50
1	1	26	F	CatPets	46						

10.3.5 Table: Token

This table contains one record per token. A token is uniquely identified by the beginning program line number and line column of the token. The token type and other columns provide secondary keys and ancillary information needed to identify certain statements token types.

Note: Only a few records have been included to show how a field statement is parsed into tokens.

FIRefNo	LnNoBeg	ColBeg	LnNoEnd	ColEnd	Token	TokenNew	TokenUpdate	TokenType	LnNo	BlkNo	FldNo	TypeNo	Keyword	KeywordOption	CCode	CVal	LnGlD
1	170	2	170	3	Q1			FName	0	1	22	41	FIEL DS	Question text		0	
1	170	4	170	4				WhiteSpaceBlank	0	1	22	41	FIEL DS	Question text		0	
1	170	5	170	8	(Q1)			FTag	0	1	22	42	FIEL DS	Question text		0	
1	170	9	170	9				WhiteSpaceBlank	0	1	22	42	FIEL DS	Question text		0	
1	170	10	170	45	"Are you ready to answer questions?"			Text	1	1	22	42	FIEL DS	Question text		0	
1	170	46	170	46	:			FDefnSeparator	0	1	22	42	FIEL DS	Question text		0	
1	170	47	170	47	(EnumBegSeparator	0	1	22	42	FIEL DS	Question text		0	
1	170	48	170	48	y			EnumCodeName	0	1	22	42	FIEL DS	ENUMERATED	y	0	
1	170	49	170	49	,			EnumSeparator	0	1	22	42	FIEL DS	ENUMERATED	y	1	
1	170	50	170	50	n			EnumCode	0	1	22	42	FIEL DS	ENUMERATED	n	1	

FIR efNo	LnNoBeg	Col Beg	LnNoEnd	Col End	Token	TokenNew	Token Update	TokenType	LnNo	BlkNo	FldNo	TypeNo	Keyword	KeywordOption	CCode	CVal	LnGlD
								eName					DS	RATED			
1	170	51	170	51)			EnumEnd Sep Separator)	0	1	22	42	FIELDS	ENUMERATED		0	

10.4 BDO Tables

Along with tables from the parser, other tables are used in the creation of the MergedBDO table. These are examples of table generated by the type definition cross reference application, the language order table, the work table language statement expansion, the table of the universe of all statement elements, and the dictionary of token types.

10.4.1 Table: ftRefTbl - Type Definition Cross Reference

The Type Definition Cross Reference table is made by the Itable application. There is one record per type number, indicating where the type is actually defined, in the end type number column. If the type number is the same as the end type number, it's a system defined data type, or the data type defined directly at the field. If the type number and end type number are different, it's a user defined data type that is defined at the type number listed in the column of type number end, i.e. the type is defined at a remote location.

Note: Types names have the same issues as fieldnames. There can be duplicate names defined in different path.

BlkNo	TypeNo	Blkpath	FName	BlkNoEnd	TypeNoEnd	TokenType
1	45	1..	B1_pet	0	0	FTName
1	46	1..	B1_pet	0	0	FTName
1	47	1..	B1_pet	0	0	FTName
1	48	1..	B2_Other	0	0	FTName
8	38	8.1..	open	8	38	FTName
8	39	8.1..	open	8	39	FTName
7	31	7.1..	open	7	31	FTName
5	22	5.1..	open	5	22	FTName
3	18	3.1..	open	3	18	FTName
1	5	1..	SERange1	1	5	TName
1	44	1..	SERange1	1	5	FTName
8	33	8.1..	string	8	33	FTName
2	11	2.1..	string	2	11	FTName
2	12	2.1..	string	2	12	FTName
1	50	1..	string	1	50	FTName
1	8	1..	T2005ToPresent	1	8	TName

BlkNo	TypeNo	Blkpath	FName	BlkNoEnd	TypeNoEnd	TokenType
1	7	1..	Tinteger	1	7	TName
8	41	8.1..	TMammals	1	25	FTName
1	25	1..	tmammals	1	25	TName
1	6	1..	Tmonth	1	6	TName
1	9	1..	Tspnspc	1	9	TName
1	3	1..	TYesNoV1	1	3	TName
8	34	8.1..	TYesNoV2	8	34	TName
8	35	8.1..	TYesNoV2	8	34	FTName
7	27	7.1..	TYesNoV2	7	27	TName
7	28	7.1..	TYesNoV2	7	27	FTName
3	14	3.1..	TYesNoV2	3	14	TName
3	15	3.1..	TYesNoV2	3	14	FTName
1	4	1..	TYesNoV2	1	4	TName
1	43	1..	TYesNoV2	1	4	FTName

10.4.2 Table: LanguageOrder

This table is used to specify the language order of the parsed datamodel, and the desired language order of the updated output files. Language reordering is rarely done. This table is updated by hand as needed. The language identifier, language order defined (parsed), and language order new definition (output) are the main columns of interest.

LngID	LngOrderDefined	LngOrderNewDefn	TextEnum	TextQuestion	TextDescriptor	TextFillEnum	TextFillQuestion	TextFillDescriptor
ENG	1	4	True	True	True	"pgm enum fill coreng"	"pgm question fill coreng"	"pgm descriptor fill coreng"
SPN	2	2	True	True	False	"pgm enum fill corspn"	"pgm question fill corspn"	"pgm descriptor fill corspn"
ALTENG	3	3	True	True	False	"pgm enum fill exteng"	"pgm question fill exteng"	"pgm descriptor fill exteng"
ALTSPN	4	1	True	True	False	"pgm enum fill extspn"	"pgm question fill extspn"	"pgm descriptor fill extspn"

10.4.3 Table: WTable – statement expansion for language tokens

This table is updated by hand. The update information in the expansion of statements, particularly in the language area, can be limited. This is done to deal with issues of scale and volume which can slow down the processing.

Note: The order_general column codes 141 is the code for the LID for question text, 142 for the question text, 161 for the LID for descriptor text, and 162 the descriptor text¹¹¹².

indexOrder	order_general	Ingno	BegMidEnd
0	130	0	1
4	142	2	0
15	150	0	2
22	161	4	0
23	162	4	0
30	170	0	3
31	310	0	0
35	331	1	0
36	332	1	0
37	331	2	0
38	332	2	0
39	331	3	0
40	332	3	0
49	340	0	4

¹¹ See the BDOVariantsUniverse Table for other code meanings.

¹² In the table the implicit space will be added for question text for the second descriptor LID and text for the fourth language, and enumeration LID and text for first three languages. The records in the table are determine by the programmer, space, and nature of the input for the update run.

10.4.4 Table: BDOVariantsUniverse – Blaise Statement token order

This table outlines all the elements and order of Blaise Statements tokens. This table should not be updated unless new statements or elements of statements have been discovered.

I've included the entirety of this table because it is a key function and analysis piece. The primary and secondary merge keys are indicated for the various statement tokens. This information was essential in writing the code and algorithms used in the merger program.

Note: The general statement order column here is the same column used in the Wtable.

ord	S D	S B	S T	S P	F F	F A	F L	F P	F T	Key y L n g	TT_Abstrac t	TT_Parsed	TT_TBDO	Order _Gen eral	Key DBT P	Key S Or F Type	Key T Num	Key F Num	Key TorF Name	Key LID	Key Cod e Name	Key Attri b	Key Defn Order	Key TT	Grp Mai n Stac k	Grp Stac k Spli t	Grp	Grp Spli t
10															merg eKey s													
20	D	B	T	P	F	A	L	P	T	Type s= =	TokenType abstract Description	(Parsed) TokenType	TokenBDO TokenType		DBT P (Bloc k Num)	Blk / Fld Type	Type Num	Field Num	Type or Field Name	Lang Ident (LID)	Code Name	Attrib Type	Defn Order	Toke nTyp e	mai n stac k grou p	spli stac k grou ping	defa ultr W/G rp	
30	y	y	y	y						*	Keyword DataModel/Bloc k/Table/Procedu re structureBegKe yword	StrctBeg	StrctBeg	21	x	s	or x	s	s					x	10	1	1	
40	y	y	y	y							name	StrctName	StrctName	22	x	s	or x	s	s					x	10	1	2	
50	y	y	y	n							desc	StrctDesc	StrctDesc	23	x	s	or x	s	s					x	10	1	3	
60	y	y	y	y							structureEndKe yword	StrctEnd	StrctEnd	900	x	s	or x	s	s					x	10	2	4	
100					y	y	y	y	y	*	Keyword	FTBeg	FTBeg	100	x	s		x	s					x	19	1		

ord	S D	S B	S T	S P	F F	F A	F L	F P	F T	K e y L i n g	TT_Abstrac t	TT_Parsed	TT_TBDO	Order _Gen eral	Key DBT P	Key S Or F Type	Key T Num	Key F Num	Key TorF Name	Key LID	Key Cod e Nam e	Key Attri b	Key Defn Orde r	Key TT	Grp Mai n Stac k	Grp Stac k Spli t	Grp	Grp Spli t
											FIELDS/AUXFI ELDS/LOCALS/ PARAMETERS/ TYPES																	
110									y	n	** Keyword Import/Export/Tr ansit	FType	FType	110	x	s		x	s					x	19	1		
120					y	y	y	y	n		FieldName	FName	FName	121	x	s	s	x1	s				x	x	19	1		
130					y	y	y	y	n		FieldName Sep ,	FNameSepa rator ,	FNameSep	122	x	s	s	x	s				x	x	19	1		
140					y	y	n	n	n		TypeTag	FTag	FTag	130	x	s	x	s	s2					x	20	1		
150					y	y	n	n	y	l	TextLID	TextLID	FTTextLID	141	x	s	x	s	s2	x			s	x	20	1		
160					y	y	n	n	y	l	Text	Text	FTText	142	x	s	x	s	s2	x			s	x	20	1		
170					y	y	n	n	n		DescSep /	DescSepara tor /	FDescSep	150	x	s	x	s	s2					x	20	1		
180					y	y	n	n	n	l	DescTextLID	DescTextLID	FDescTextLID	161	x	s	x	s	s2	x			s	x	20	1		
190					y	y	n	n	n	l	DescText	DescText	FDescText	162	x	s	x	s	s2	x			s	x	20	1		
200					y	y	y	y	n		Type Sep :	FDefnSep Separator :	FDefnSep	170	x	s	x	s	s2					x	20	1		
210	y				n	n	n	n	n		Languages	LANGUAGE S	FTBeg	25	x	s	x		s2					x				
220	y				n	n	n	n	y		TypeNameDef	Tname	TName	26	x	s	x	s	s2					x	20	1		
230	y				n	n	n	n	y		Type =	TNameSep Separator =	TNameSep	28	x	s	x	s	s2					x	20	1		
240	y				n	n	n	n	y			LANGUAGE SSeparator		27	x	s	x	s	s2					x	20	1		

ord	S D	S B	S T	S P	F F	F A	F L	F P	F T	Key L N g	TT_Abstrac t	TT_Parsed	TT_TBDO	Order _Gen eral	Key DBT P	Key S Or F Type	Key T Num	Key F Num	Key TorF Name	Key LID	Key Cod e Nam e	Key Attri b	Key Defn Orde r	Key TT	Grp Mai n Stac k	Grp Stac k Spli t	Grp	Grp Spli t
											=																	
250																												
260					y	y	n	y	y	*	Keyword Array/Set	ASBeg	ASBeg	200	x	s	x	s	s2				x	20	1	B		
270					y	y	n	y	y		[ASRngBegS eparator [ASRngBeg	210	x	s	x	s	s2				x	20	1	B		
280					y	y	n	y	y		min	RngValBeg	ASRngValBeg	220	x	s	x	s	s2			x	x	20	1	B		
290					y	y	n	y	y	*	..	Separator ..	ASRngValSep	230	x	s	x	s	s2			x	x	20	1	B	B2	
300					y	y	n	y	y		max	RngValEnd	ASRngValEnd	240	x	s	x	s	s2			x	x	20	1	B	B2	
310					y	y	n	y	y		sep ,	Separator ,	ASRngValSet Sep	241	x	s	x	s	s2			x	x	20	1	B	B2	
320					y	y	n	y	y]	ASRngEndS eparator]	ASRngEnd	250	x	s	x	s	s2				x	20	2	B	B2	
330					y	y	n	y	y		of	ASSepOF	ASSep	260	x	s	x	s	s2				x	20	1	B		
340																												
350																												
360					y	y	n	y	y	*	EnumBeg(EnumBeg Separator (EnumBeg Separator (EnumBeg	300	x	s	x	s	s2				x	20	1	C	1	
370					y	y	n	y	y		EnumCodeNam e	EnumCodeN ame	EnumCodeNa me	310	x	s	x	s	s2		x		s	x	21	1	C	2
380					y	y	n	y	y		EnumCodeNum Beg(EnumCodeNum BegSepS eparator (EnumCodeN umBegSepS eparator (EnumCodeNu mBeg	321	x	s	x	s	s2		x		s	x	21	1	C	2
390					y	y	n	y	y		EnumCodeNum um	EnumCodeN um	EnumCodeNu m	322	x	s	x	s	s2		x		s	x	21	1	C	2
400					y	y	n	y	y		EnumCodeNum End) EnumCodeN umEndSepS eparator)	EnumCodeN umEndSepS eparator)	EnumCodeNu mEnd	323	x	s	x	s	s2		x		s	x	21	1	C	2
410																												

ord	S D	S B	S T	S P	F F	F A	F L	F P	F T	Key L N g	TT_Abstrac t	TT_Parsed	TT_TBDO	Order _Gen eral	Key DBT P	Key S Or F Type	Key T Num	Key F Num	Key TorF Name	Key LID	Key Cod e Nam e	Key Attri b	Key Defn Orde r	Key TT	Grp Mai n Stac k	Grp Stac k Spli t	Grp	Grp Spli t
420					y	y	n	y	y	l	EnumDesTextLID	EnumDescTextLID	EnumDesTextLID	331	x	s	x	s	s2	x	x		s	x	21	1	C	2
430					y	y	n	y	y	l	EnumDesText	EnumDescText	EnumDesText	332	x	s	x	s	s2	x	x		s	x	21	1	C	2
440																												
450					y	y	n	y	y		EnumCodeSep	EnumSepSeparator	EnumCodeSep	340	x	s	x	s	s2		x		s	x	21	1	C	2
460					y	y	n	y	y		EnumEnd)	EnumEndSep Separator)	EnumEnd	350	x	s	x	s	s2				x		20	2	C	1
470																												
480																												
490					y	y	y	y	y	*	TypeNameRef (system or user defn)	FTName	FTName	400	x	s	x	s	s2				x		20	1	D	1
500					y	y	n	y	y		[Separator [FTRngBeg	410	x	s	x	s	s2				x		20	1	D	2
510					y	y	n	y	y		min	RngValBeg	FTRngValBeg	420	x	s	x	s	s2				x		20	1	D	2
520					y	y	n	y	y	*	..	Separator ..	FTRngValSep	430	x	s	x	s	s2				x		20	1	D	3
530					y	y	n	y	y		max	RngValEnd	FTRngValEnd	440	x	s	x	s	s2				x		20	1	D	3
540					y	y	n	y	y		sep ,	Separator ,	FTRngValSet Sep	441	x	s	x	s	s2			x	x	20	1	B	B2	
550					y	y	n	y	y]	Separator]	FTRngEnd	450	x	s	x	s	s2				x		20	2	D	2
560																												
570																												
580	y	n	n	n	y	y	n	y	y	**	Keyword ATTRIBUTES	ATTR	ATTR	600	x	s						x	x			1		0
590	y	n	n	n	y	y	n	y	y		ATTRIBUTES =	ATTRSepSeparator =	ATTRSep	601	x	s						x	x			1		0
600	y	n	n	n	y	y	n	y	y	*	sep ,	ATTRIBDRE SepSeparato		602	x	s		s				x	x	x		1		1

ord	S D	S B	S T	S P	F F	F A	F L	F P	F T	Key L i n g	TT_Abstrac t	TT_Parsed	TT_TBDO	Order _Gen eral	Key DBT P	Key S Or F Type	Key T Num	Key F Num	Key TorF Name	Key LID	Key Cod e Nam e	Key Attri b	Key Defn Order	Key TT	Grp Mai n Stac k	Grp Stac k Spli t	Grp	Grp Spli t
												r ,																
610	y	n	n	n	y	y	n	y	y		toggle type(DK/NDK or RF/RF or Empty or non-Empty)			610	x	s	x	s	s2			x	x	x		1	1	
620																												
630	y	n	n	n	y	y	n	y	y		attrib sep ,	ATTRIBDRE SepSeparato r ,	ATTRDSEP	611		s							x					
640	y	n	n	n	y	y	n	y	y		DKNDK sep ,	ATTRIBDRE SepSeparato r ,	ATTRDSEP	612		s							x					
650	y	n	n	n	y	y	n	y	y		DKNDK	ATTRDK	ATTRD	613		s							x					
670	y	n	n	n	y	y	n	y	y		RFNRF sep ,	ATTRIBDRE SepSeparato r ,	ATTRRSEP	622		s							x					
680	y	n	n	n	y	y	n	y	y		RFNRF	ATTRNRF	ATTRR	623		s							x					
700	y	n	n	n	y	y	n	y	y		EMPTYNEMPT Y sep ,	ATTRIBDRE SepSeparato r ,	ATTRESEP	632		s							x					
710	y	n	n	n	y	y	n	y	y		EMPTYNEMPT Y	ATTRE	ATTRE	633		s							x					
660	y	n	n	n	y	y	n	y	y		NDK	ATTRNDK	ATTRNDK	614		s							x					
690	y	n	n	n	y	y	n	y	y		NRF	ATTRRF	ATTRNRF	624		s							x					
720	y	n	n	n	y	y	n	y	y		EMPTYNEMPT Y	ATTRNE	ATTRNE	634		s							x					
760											Comment	COMMENT	Comment	11	x								x	x	30	1		
770											WhiteSpaceBl ank	WhiteSpace Blank	WhiteSpaceBl ank	12	x								x	x	80	1		
780											WhiteSpaceTab	WhiteSpace	WhiteSpaceTa	13	x								x	x	90	1		

ord	S D	S B	S T	S P	F F	F A	F L	F P	F T	Key L N g	TT_Abstrac t	TT_Parsed	TT_TBDO	Order _Gen eral	Key DBT P	Key S Or F Type	Key T Num	Key F Num	Key TorF Name	Key LID	Key Cod e Nam e	Key Attri b	Key Defn Orde r	Key TT	Grp Mai n Stac k	Grp Stac k Spli t	Grp	Grp Spli t
												Tab	b															
790																												
800										*	Keyword Router	ROUTER		40											40	1		
810											Name			41											40	1		
820										*	Keyword Alien	ALIEN		45	x								x		40	2		
830											(Alien Separator (46											40	2		
860											MethodName DLLFileName	Alien second		49											40	2		
880											DLLProcName DLLProcNumbe r	Alien third		51											40	2		
900																												
910																												
920																												
930										*	Keyword Include (key word)	INCLUDE		31	x								x	x	50	1		
940											Include file name	File Name	File Name	32	x								x	x	50	1		
950																												
960										*	Keyword Parallel	PARALLEL		71	x								x	x	60	1		
970											name			72	x		x						x	x	60	1		
980											=			73	x		x						x	x	60	1		
990											Field			74	x		x						x	x	60	1		
1000																												
1010										*	Keyword Primary/Second	PRIMARY		61	x		x							x	70	1		

ord	S D	S B	S T	S P	F F	F A	F L	F P	F T	Key y L n g	TT_Abstrac t	TT_Parsed	TT_TBDO	Order _Gen eral	Key DBT P	Key S Or F Type	Key T Num	Key F Num	Key TorF Name	Key LID	Key Cod e Nam e	Key Attri b	Key Defn Orde r	Key TT	Grp Mai n Stac k	Grp Stac k Spli t	Grp	Grp Spli t
											ary																	
1020											field name	Name		62	x			x1					x	x	70	1		
1030											comma	Separator ,		63	x			x					x	x	70	1		
1040										*	Keyword Primary/Second ary	SECONDAR Y		65														
1050											field name	Name		66														
1060											comma	Separator ,		67														
1070																												
1080	y	y	y	y							Keyword Rules			801														
1090	y	y	y	y							Rules Text	RulesBlock		800														
1104												text-double quote		4														
1110												text-single quote		3														
1120												Nonwhite- delim		2														
1140												Separator ;		10														
1130												Nonwhite		1														
1150												KeyWord		0														
1160																												
1170										*	- triggers/controls BDO records																	
1180										x	-primary keys																	
1190										s	- secondary keys																	
1200	D	B	T	P	F	A	L	P	T	F	Keyword values from Token Region																	

ord	S D	S B	S T	S P	F F	F A	F L	F P	F T	Key y L n g	TT_Abstrac t	TT_Parsed	TT_TBDO	Order _Gen eral	Key DBT P	Key S Or F Type	Key T Num	Key F Num	Key TorF Name	Key LID	Key Cod e Nam e	Key Attri b	Key Defn Orde r	Key TT	Grp Mai n Stac k	Grp Stac k Spli t	Grp	Grp Spli t
									e s =		Fields(FALPFTy pes)Rules																	
1210											ALIEN																	
1220											ATTRIBUTES																	
1230						x					AUXFIELDS																	
1240	x										BLOCK																	
1250	x										DATAMODEL																	
1260	x	1	1								ENDBLOCK																	
1270	x										ENDMODEL																	
1280		1	1	x							ENDPROCEDU RE																	
1290		1	x	1							ENDTABLE																	
1300					x						FIELDS																	
1310											INCLUDE																	
1320									1		LANGUAGES																	
1330						x					LOCALS																	
1340								x			PARAMETERS																	
1350								x			PARAMETERS EXPORT																	
1360								x			PARAMETERS IMPORT																	
1370								x			PARAMETERS TRANSIT																	
1380											PRIMARY																	
1390				x							PROCEDURE																	
1400											ROUTER																	
1410											RULES																	

ord	S D	S B	S T	S P	F F	F A	F L	F P	F T	Key L n g	TT_Abstrac	TT_Parsed	TT_TBDO	Order _Gen eral	Key DBT P	Key S Or F Type	Key T Num	Key F Num	Key TorF Name	Key LID	Key Cod e Nam e	Key Attri b	Key Defn Orde r	Key TT	Grp Mai n Stac k	Grp Stac k Spli t	Grp	Grp Spli t
1420											SECONDARY																	
1430			x								TABLE																	
1440									x		TYPE																	
1450																												
1460																												
1470											PARALLEL																	
1480											USES																	
1490											EXTERNAL	EXTERNAL S Meta Information																
1500												EXTERNAL S reference name																
1510																												
1520																												
1530																												
1540																												
730																												
740																												
750																												
840											ProgID	ProgID		47											40	2		
850											,	Alien secondSepa rator ,		48											40	2		
890)	Alien Separator)		52											40	2		
870											,	Alien thirdSeparat or ,		50											40	2		

ord	S D	S B	S T	S P	F F	F A	F L	F P	F T	Key y L n g	TT_Abstrac t	TT_Parsed	TT_TBDO	Order _Gen eral	Key DBT P	Key S Or F Type	Key T Num	Key F Num	Key TorF Name	Key LID	Key Cod e Nam e	Key Attri b	Key Defn Orde r	Key TT	Grp Mai n Stac k	Grp Stac k Spli t	Grp	Grp Spl it
1100																												
1102												KeyWordLog ic		5														

10.4.5 Table: SE_DatabaseDictionary – A Dictionary of Token Types

This table is used to determine the names of the token types used in the MergedBDO. It is through these names that the token types have more meaning than the token types described by the parser.

Note: TokenTypeID corresponds to Order_General in the table BDOVariantsUniverse. In this dictionary these same abstracted token types would have one or more token types depending on what type of structure it is in.

Note: Records in this table are reduced.

TokenTypeID	D	B	TB L	P	Fld	Aux	Local	ParmFld	TYP	TokenType	Definition	Restrictions	Mandatory	Keyword
11	-	-	-	-	-	-	-	-	-	Comment	Text enclosed by braces not evaluated by the compiler includes { }		False	False
21	y				-	-	-	-	-	Datamodel	Declares and names a data model.		True	True
21		y			-	-	-	-	-	Block	Defines a type as a block: a user defined sub model of related fields and rules		True	True
22		y			-	-	-	-	-	BlockName	Name of block		True	False
100	-	-	-	-		y				Auxfields	Defines auxiliary fields to store temporary information, only available during the application.		False	True
100	-	-	-	-	y					Fields	indicates the beginning of a fields section. In this section you define one or more fields in the current block.		False	True
142	-	-	-	-	y	y	n	n	y	FieldText	Text related to a Field		False	False
310	-	-	-	-	y	y	n	y	y	CodeName	Defines the domain of a type as a discrete range in which each category is specified as an identifier. Each category identifier may have a code number and a text		False	False
141	-	-	-	-	y	y	n	n	y	FieldTextLangID			False	False
161	-	-	-	-	y	y	n	n	n	DescTextLangID			False	False

10.5 Procedures: Showing Key Relationships between BDO Tables

Putting the SQL of procedures is a concise and cryptic way to share primary and secondary keys used in the table for creating the BDO, and how these pieces get put together. The SQL statements provide clues to entity relationships between tables. If a table or procedure name referenced by the procedure does not appear here, the chart at the beginning of the appendix will provide a brief description of it.

10.5.1 Procedure: SEb01 – populates ITTemplate with records

```
INSERT INTO dbo.ITTemplate
    (BlkNo, TypeNo, Blkpath, Name, TokenType, FldNo)
SELECT  dbo.Token.BlkNo, dbo.Token.TypeNo, dbo.Blocks.Blkpath,
dbo.Token.Token AS Name, dbo.Token.TokenType, dbo.Token.FldNo
FROM    dbo.Token LEFT OUTER JOIN
        dbo.Blocks ON dbo.Token.BlkNo = dbo.Blocks.BlkNo
WHERE   (dbo.Token.TokenType = N'FTName') OR
        (dbo.Token.TokenType = N'TName') OR
        (dbo.Token.TokenType = N'StrctName')
```

10.5.2 Procedure: SEbft01 – populates FT with records from ITTemplate

```
INSERT INTO dbo.ft
    (BlkNo, TypeNo, FldNo, TName)
SELECT  BlkNo, TypeNo, FldNo, Name AS TName
FROM    dbo.ITTemplate
WHERE   (TokenType = N'TName') OR
        (TokenType = N'StrctName')
ORDER BY BlkNo DESC, TypeNo DESC
```

10.5.3 Procedure: MakeTokenBDOPrep – Makes a combine view of several tables

```
SELECT  TOP 100 PERCENT dbo.Token.LnNoBeg, dbo.Token.ColBeg,
dbo.Token.LnNoEnd, dbo.Token.ColEnd, dbo.Token.BlkNo, dbo.Token.FldNo,
        dbo.Token.TypeNo, dbo.Token.CCode, dbo.Token.LngID,
dbo.Token.TokenType, dbo.Token.Keyword, dbo.Token.Token, dbo.Token.TokenNew,
        dbo.Token.TokenUpdate, dbo.Token.KeywordOption, dbo.Token.LngNo,
dbo.Token.CVal, dbo.Token.FlRefNo, dbo.FieldType.FieldType,
        dbo.Blocks.Blktype, dbo.Token.FldNo AS FldNoRng,
dbo.BDOVariantsUniverseRoll.Order_General, dbo.Token.TokenType AS
TokenTypeBDO
FROM    dbo.Token LEFT OUTER JOIN
        dbo.BDOVariantsUniverseRoll ON dbo.Token.TokenType =
dbo.BDOVariantsUniverseRoll.TT_Parsed LEFT OUTER JOIN
        dbo.FieldType ON dbo.Token.FldNo = dbo.FieldType.FieldNo AND
dbo.Token.BlkNo = dbo.FieldType.BlkNo LEFT OUTER JOIN
        dbo.Blocks ON dbo.Token.BlkNo = dbo.Blocks.BlkNo
ORDER BY dbo.Token.LnNoBeg, dbo.Token.ColBeg
```

10.5.4 Procedure: make table tta – backup table where token types get revised

```
SELECT TOP 100 PERCENT dbo.TokenBDOTest.TokenID,
dbo.TokenBDOTest.LnNoBeg, dbo.TokenBDOTest.ColBeg,
dbo.TokenBDOTest.LnNoEnd,
    dbo.TokenBDOTest.ColEnd, dbo.TokenBDOTest.BlkNo,
dbo.TokenBDOTest.FldNo, dbo.TokenBDOTest.TypeNo, dbo.TokenBDOTest.CCode,
    dbo.TokenBDOTest.Keyword, dbo.TokenBDOTest.Token,
dbo.TokenBDOTest.TokenNew, dbo.TokenBDOTest.TokenUpdate,
    dbo.TokenBDOTest.KeywordOption, dbo.TokenBDOTest.LngNo,
dbo.TokenBDOTest.CVal, dbo.TokenBDOTest.FIRefNo,
    dbo.TokenBDOTest.FldNo AS FldNoRng,
dbo.TokenBDOTest.Order_General, dbo.BDOVariantsUniverseRoll.TT_Parsed AS
TokenType,
    dbo.LanguageOrder.LngID, dbo.TokenBDOTest.Blktype,
dbo.TokenBDOTest.TokenTypeBDO, dbo.TokenBDOTest.FieldType,
    dbo.TokenBDOTest.DupFiles
INTO     dbo.tta
FROM     dbo.TokenBDOTest INNER JOIN
    dbo.BDOVariantsUniverseRoll ON dbo.TokenBDOTest.Order_General =
dbo.BDOVariantsUniverseRoll.Order_General LEFT OUTER JOIN
    dbo.LanguageOrder ON dbo.TokenBDOTest.LngOrderNewDefn =
dbo.LanguageOrder.LngOrderNewDefn
ORDER BY dbo.TokenBDOTest.TokenID, dbo.TokenBDOTest.LnNoBeg,
dbo.TokenBDOTest.ColBeg
```

10.5.5 Procedure: MakeMergedBDO – Makes the table used by the merger

```
SELECT TOP 100 PERCENT dbo.tta.TokenID, dbo.tta.TokenTypeBDO AS
TokenType, dbo.tta.Token, dbo.tta.BlkNo AS BlockNum,
    dbo.Blocks.Blkname AS BlockName, dbo.tta.FldNo AS FieldNum,
dbo.FieldType.FieldName, dbo.tta.TypeNo AS TypeNum, dbo.tta.LngID AS LangID,
    dbo.tta.CCode AS CodeValue, dbo.tta.LnNoBeg AS LineNumBeg,
dbo.tta.ColBeg AS ColumnNumBeg, dbo.tta.LnNoEnd AS LineNumEnd,
    dbo.tta.ColEnd AS ColumnNumEnd, dbo.tta.FIRefNo AS FileRefNum,
dbo.Files.FIName AS FileName, dbo.tta.TokenUpdate,
    dbo.Blocks.Blkpath AS BlockPath, dbo.FileLines.FILnCnt AS
FileLineCountBeg, FileLines_1.FILnCnt AS FileLineCountEnd,
dbo.ftRefTblRollup.BlkNoEnd,
    dbo.ftRefTblRollup.TypeNoEnd, dbo.TypeOF.TypeName,
dbo.DupFilesB.FIName, dbo.DupFilesA10.[count] AS FileUseCount,
    dbo.Files.FILnCnt AS FileLineNumMax
INTO     dbo.MergedBDO
FROM     dbo.FileLines RIGHT OUTER JOIN
    dbo.DupFilesB INNER JOIN
    dbo.DupFilesA10 ON dbo.DupFilesB.FIName = dbo.DupFilesA10.FIName
RIGHT OUTER JOIN
    dbo.Files ON dbo.DupFilesB.FIRefNo = dbo.Files.FIRefNo RIGHT
OUTER JOIN
    dbo.ftRefTblRollup LEFT OUTER JOIN
```

```

        dbo.TypeOF ON dbo.ftRefTblRollup.TypeNoEnd = dbo.TypeOF.TypeNo
RIGHT OUTER JOIN
        dbo.tta ON dbo.ftRefTblRollup.TypeNo = dbo.tta.TypeNo AND
dbo.ftRefTblRollup.BlkNo = dbo.tta.BlkNo LEFT OUTER JOIN
        dbo.FieldType ON dbo.tta.BlkNo = dbo.FieldType.BlkNo AND
dbo.tta.FldNo = dbo.FieldType.FieldNo LEFT OUTER JOIN
        dbo.FileLines FileLines_1 ON dbo.tta.FIRefNo = FileLines_1.FIRefNo
AND dbo.tta.LnNoEnd = FileLines_1.LnNo ON
        dbo.Files.FIRefNo = dbo.tta.FIRefNo LEFT OUTER JOIN
        dbo.Blocks ON dbo.tta.BlkNo = dbo.Blocks.BlkNo ON
dbo.FileLines.FIRefNo = dbo.tta.FIRefNo AND dbo.FileLines.LnNo = dbo.tta.LnNoBeg
ORDER BY dbo.tta.TokenID, dbo.tta.LnNoBeg, dbo.tta.ColBeg, dbo.tta.LnNoEnd,
dbo.tta.ColEnd, dbo.tta.LngID

```

10.5.6 Procedure: SE0_makeFromTest

```

SELECT TOP 100 PERCENT dbo.TokenBDOTest.TokenID,
dbo.TokenBDOTest.LnNoBeg, dbo.TokenBDOTest.ColBeg,
dbo.TokenBDOTest.LnNoEnd,
        dbo.TokenBDOTest.ColEnd, dbo.TokenBDOTest.BlkNo,
dbo.TokenBDOTest.FldNo, dbo.TokenBDOTest.TypeNo, dbo.TokenBDOTest.CCode,
        dbo.TokenBDOTest.Keyword, dbo.TokenBDOTest.Token,
dbo.TokenBDOTest.TokenNew, dbo.TokenBDOTest.TokenUpdate,
        dbo.TokenBDOTest.KeywordOption, dbo.TokenBDOTest.LngNo,
dbo.TokenBDOTest.CVal, dbo.TokenBDOTest.FIRefNo,
        dbo.TokenBDOTest.FldNo AS FldNoRng,
dbo.TokenBDOTest.Order_General, dbo.BDOVariantsUniverseRoll.TT_Parsed AS
TokenType,
        dbo.LanguageOrder.LngID, dbo.TokenBDOTest.Blktype,
dbo.TokenBDOTest.TokenTypeBDO, dbo.TokenBDOTest.FieldType,
        dbo.TokenBDOTest.DupFiles
INTO      dbo.tta
FROM      dbo.TokenBDOTest INNER JOIN
        dbo.BDOVariantsUniverseRoll ON dbo.TokenBDOTest.Order_General =
dbo.BDOVariantsUniverseRoll.Order_General LEFT OUTER JOIN
        dbo.LanguageOrder ON dbo.TokenBDOTest.LngOrderNewDefn =
dbo.LanguageOrder.LngOrderNewDefn
ORDER BY dbo.TokenBDOTest.TokenID, dbo.TokenBDOTest.LnNoBeg,
dbo.TokenBDOTest.ColBeg

```

10.6 Merger Tables

10.6.1 User Input Format

This is table used by the merger to get update records from. The translator/preprocessor writes all user input record to this one table for processing by the merger. Different token types can have different merge keys. Certain columns will be populated or unpopulated based on the token type. This table was mentioned in the paper. However all of the columns are not, nor do the column heading give a clear clue as to what or how it's used for. Since this is an important table in the design of the merger application I'm listing

tersely what each column or theme of columns is supposed to do instead of showing an example of table records.

Column Name Description

TokenType - as described by the dictionary

TokenTypeParser- the token type as described by the parser (This is a place holder for merging directly in to the token table if a BDO expansion isn't needed.)

Token - the revise/update data for the token

TokenUpdate - delete or update the token

BlockName - The defined block name

BlockNameOriginal – one application we use will append “_2” to a duplicate block name. This is a space holder to preserve this variant of the name.

BlockNum - When a block name appears more than once, you need to put in the actual block number to allow for a unique merge.

OvrUseBlkNum- When needing to use the BlockNum to process an update, you need to let the merger know, by putting data in this column.

FieldNameOriginal (same theme as duplicate blocks)

FieldName

FieldNum

TypeNameOriginal (same theme as duplicate blocks)

TypeName

TypeNum

OvrUseTypNum

FieldType

CodeValue

LangID

OvrUseAllLanguages – you want to use the same token to update all languages in the same manner for a token type

TokenProcessed - feedback from the merger saying if the token was found, or updated

TokenAcceptRejectCode

TokenAcceptRejectText

FileRefNum (these are planned to be used when you have the same include file used more than once)

OvrUseFileNum

Cheapkey – a cheap key added to uniquely identify a user update request record

BulkID – a duplicate of cheap key

BulkSource – since update records can come from several difference sources, the pre-processor will put something indicating the group or file the record came from.

10.6.2 Merged BDO

These are some records from the merge BDO table used by the merger and writer.

Note: Empty line information does not appear in this table. The writing routine compares line numbers between records and puts in the blank lines as needed.

Token ID	TokenType	Token	Block Num	BlockName	Field Num	FieldName	Type Num	Lang ID	Code Value	Line Num Beg	Column Num Beg	Line Num End	Column Num End	File Ref Num	FileName	Token Update	Block Path	File Line Count Beg	File Line Count End	Blk No End	Type No End	Type Name	FIName	File Use Count	File Line Max	chekey
1246	FieldName	Q1	1	SE_illustration	22	Q1	41			170	2	170	3	1	SE_Illustration.bla		1..	58	58	0				99	1246	
1247	WhiteSpaceBlank		1	SE_illustration	22	Q1	41			170	4	170	4	1	SE_Illustration.bla		1..	58	58	0				99	1247	
1248	Tag	(Q1.1)	1	SE_illustration	22	Q1	42			170	5	170	8	1	SE_Illustration.bla	Update	1..	58	58	0				99	1248	
1249	WhiteSpaceBlank		1	SE_illustration	22	Q1	42			170	9	170	9	1	SE_Illustration.bla		1..	58	58	0				99	1249	
1250	FieldTextLangID		1	SE_illustration	22	Q1	42	ENG		9				1	SE_Illustration.bla		1..			0				99	1250	
1251	FieldText	"Are you ready to answer questions?"	1	SE_illustration	22	Q1	42	ENG		170	10	170	45	1	SE_Illustration.bla		1..	58	58	0				99	1251	
1252	FieldTextLangID		1	SE_illustration	22	Q1	42	SPN		10				1	SE_Illustration.bla		1..			0				99	1252	

MergedBDO

Token ID	TokenType	Token	BlockNum	BlockName	FieldNum	FieldName	TypeNum	LangID	CodeValue	LineNumBeg	ColumnNumBeg	LineNumEnd	ColumnNumEnd	FileRefNum	FileName	TokenUpdate	BlockPath	FileLineCountBeg	FileLineCountEnd	BlkNoEnd	TypeNoEnd	TypeName	FIName	FileUseCount	FileLineNumMax	cheap key
1258	Separator/		1	SE_illustration	22	Q1	42				10			1	SE_Illustration.bla		1..				0				99	1258
1259	DescTextLangID		1	SE_illustration	22	Q1	42	ALTS PN			10			1	SE_Illustration.bla		1..				0				99	1259
1260	FieldDescriptor		1	SE_illustration	22	Q1	42	ALTS PN			10			1	SE_Illustration.bla		1..				0				99	1260
1261	Separator:	:	1	SE_illustration	22	Q1	42			170	46	170	46	1	SE_Illustration.bla		1..	58	58		0				99	1261
1262	Separator_Enum	(1	SE_illustration	22	Q1	42			170	47	170	47	1	SE_Illustration.bla		1..	58	58		0				99	1262
1263	CodeName	y	1	SE_illustration	22	Q1	42		y	170	48	170	48	1	SE_Illustration.bla		1..	58	58		0				99	1263
1264	CodeTextLangID		1	SE_illustration	22	Q1	42	ENG	y		48			1	SE_Illustration.bla		1..				0				99	1264

MergedBDO

TokenID	TokenType	Token	BlockNum	BlockName	FieldNum	FieldName	TypeNum	LangID	CodeValue	LineNumBeg	ColumnNumBeg	LineNumEnd	ColumnNumEnd	FileRefNum	FileName	TokenUpdate	BlockPath	FileLineBeg	FileLineEnd	BlkNoEnd	TypeNoEnd	TypeName	FileName	FileUseCount	FileLineMax	checkkey
47	TYPE	type	1	SE_illustration	0	SE_illustration	2			13	1	13	4	2	type_Definitions.INC		1..	2	2		0				34	47
48	WhiteSpaceBlank		1	SE_illustration	0	SE_illustration	2			14	1	14	8	2	type_Definitions.INC		1..	3	3		0				34	48
49	Comment	{V1..}	1	SE_illustration	0	SE_illustration	2			14	9	14	87	2	type_Definitions.INC		1..	3	3		0				34	49
50	WhiteSpaceTab		1	SE_illustration	0	SE_illustration	2			15	1	15	1	2	type_Definitions.INC		1..	4	4		0				34	50
51	TypeName	TypesNoV1	1	SE_illustration	0	SE_illustration	3			15	2	15	9	2	type_Definitions.INC		1..	4	4	1	3	TypesNoV1			34	51
52	WhiteSpaceBlank		1	SE_illustration	0	SE_illustration	3			15	10	15	10	2	type_Definitions.INC		1..	4	4	1	3	TypesNoV1			34	52
53	Separator=	=	1	SE_illustration	0	SE_illustration	3			15	11	15	11	2	type_Definitions.INC		1..	4	4	1	3	TypesNoV1			34	53
54	Separator(Enum)	(1	SE_illustration	0	SE_illustration	3			15	12	15	12	2	type_Definitions.INC		1..	4	4	1	3	TypesNoV1			34	54
55	WhiteSpaceBlank		1	SE_illustration	0	SE_illustration	3			15	13	15	13	2	type_Definitions.INC		1..	4	4	1	3	TypesNoV1			34	55
56	CodeName	Y	1	SE_illustration	0	SE_illustration	3		Y	15	14	15	14	2	type_Definitions.INC		1..	4	4	1	3	TypesNoV1			34	56
57	WhiteSpaceBlank		1	SE_illustration	0	SE_illustration	3		Y	15	15	15	15	2	type_Definitions.INC		1..	4	4	1	3	TypesNoV1			34	57

MergedBDO

TokenID	TokenType	Token	BlockNum	BlockName	FieldNum	FieldName	TypeNum	LangID	CodeValue	LineNumBeg	ColumnNumBeg	LineNumEnd	ColumnNumEnd	FileRefNum	FileName	TokenUpdate	BlockPath	FileLineCountBeg	FileLineCountEnd	BlkNoEnd	TypeNoEnd	TypeName	FileName	FileUseCount	FileLineNumMax	checkkey
58	Separator_code num	(1	SE_illustration	0	SE_illustration	3		Y	15	16	15	16	2	type_Definitions.INC	Delete	1..	4	4	1	3	TYesNoV1			34	58
59	CodeNumber		2	SE_illustration	0	SE_illustration	3		Y	15	17	15	17	2	type_Definitions.INC	Delete	1..	4	4	1	3	TYesNoV1			34	59
60	Separator_code num)	1	SE_illustration	0	SE_illustration	3		Y	15	18	15	18	2	type_Definitions.INC	Delete	1..	4	4	1	3	TYesNoV1			34	60
61	WhiteSpaceBlank		1	SE_illustration	0	SE_illustration	3		Y	15	19	15	19	2	type_Definitions.INC		1..	4	4	1	3	TYesNoV1			34	61
62	CodeTextLangID		1	SE_illustration	0	SE_illustration	3	ENG	Y		19			2	type_Definitions.INC		1..			1	3	TYesNoV1			34	62
63	CodeText	"yes"	1	SE_illustration	0	SE_illustration	3	ENG	Y	15	20	15	24	2	type_Definitions.INC		1..	4	4	1	3	TYesNoV1			34	63

10.7 Merger Log Information

10.7.1 The Check Option Report

SE Merger Application

Run 07/23/2007 6:39:45 PM

Check User Input Versus Blaise DataModel Structure

Language Defined and Order

Run 07/23/2007 6:39:45 PM

- 1.CORENG (old order 1)
- 2.CORSPN (old order 2)
- 3.PRXENG (old order 3)
- 4.PRXSPN (old order 4)
- 5.EXTENG (old order 5)
- 6.EXTSPN (old order 6)
- 7.MEDIA (old order 7)

BDO language Expansion Anchors

Run 07/23/2007 6:39:45 PM

>>>>> For Field Text LIDs, there are no Merged BDO anchors.
ok -- For Field Text, all of the possible 7 Merged BDO anchors.

>>>>> For Field Descriptor Text LIDs, there are no Merged BDO anchors.
>>>>> For Field Descriptor Text, there are only 1 of the possible 7 Merged BDO anchors.

>>>>> For CodeFrame Text LIDs, there are no Merged BDO anchors.
ok -- For CodeFrame Text, all of the possible 7 Merged BDO anchors.

Duplicate Blocks and Include file usage

Run 07/23/2007 6:39:45 PM

Duplicate Blocks Information

>>>>> The following are 'duplicate' block name.
>>>>> The user input file will need to use Block numbers to uniquely distingusih these blocks.

The Block Name BA_WThree appears 2 times.
The Block Name BF_DECISIONS appears 2 times.

Duplicate (re-used)Include File Information

>>>>> The following are 'duplicate' include file name.
>>>>> The user input updates to these file reference numbers - may conflict with each other or not be updated.

The File Name HRS06_W1.INC appears 2 times.

10.7.2 The Merger Log

SE Merger Application

Run 07/23/2007 6:44:07 PM
Checking Source Editor Database Connection

07/23/2007 6:44:07 PM
Server Connection Okay 07/23/2007 6:44:07 PM

Statistic of user input table

07/23/2007 6:44:07 PM
Getting User Input 07/23/2007 6:44:07 PM
There are N=6498 User Input TokenType update request records.

331E-Code Frame Enumeration Language Dependent Structure Keys: TokenType, Block Name, Type Name, CodeValue, and LangID.

>>>>> (51)>>>>> The user input record was not in the DataModel BDO being updated.

TokenType: CodeText
BlockName: B_EMPLOYERINFO
FieldName: W168_VerifyEmpName
TypeName : TW168_VerifyEmpName
CodeValue: DENIESWRKPW
LangID : prxspn

331F-Code Frame Field Language Dependent Structure Keys: TokenType, Block Name, Field Name, CodeValue, and LangID.

>>>>> (51)>>>>> The user input record was not in the DataModel BDO being updated.

TokenType: CodeText
BlockName: BM_SSDI_APPACCEPTED
FieldName: W256A
TypeName :
CodeValue: RETURNEDWORK
LangID : prxspn

141/161-Field Language Dependent Structure Keys: TokenType, Block Name, Field Name, and LangID.

>>>>> (51)>>>>> The user input record was not in the DataModel BDO being updated.

TokenType: FieldText
BlockName:
FieldName:
TypeName :
CodeValue:
LangID : PRXSPN

141/161-Field Language Dependent Structure Keys: TokenType, Block Name, Field Name, and LangID.

>>>>> (51)>>>>> The user input record was not in the DataModel BDO being updated.

TokenType: FieldText
BlockName: B_EMPLOYERINFO
FieldName: W158_CurrEmpName
TypeName :
CodeValue:
LangID : PRXSPN

... now processing TokenTypes of CodeText. with record 2967 of a total of 6498.
45.6602031394275% processed. 07/23/2007 6:44:58 PM

10.8 A Sample Output file

The output has several changes due to relative positing and the language re-order done for this run.

10.8.1 FileName: PetFields.inc

```
{field for the pet blocks}
type TYesNoV2 =      ( Y (11) {SE Added text} "" {SE Added text} "" {SE Added text} "" "yes",
                    N {SE Added text} "" {SE Added text} "" {SE Added text} "" "no",
                    s {SE Added text} "" {SE Added text} "" {SE Added text} "" "Several"), dk
```

```
fields TP1
    {SE Added text}""
```

```
{SE Added text}""
```

```
{SE Added text}""
eng "For this type of pet, ^pettype, @/
@/Do you have any? " alteng"For this type of pet, ^pettype, @/
@/Do you have any? ":TYesNoV2
TP2 "How many ^pettype do you have?": 0..100
tp3 (Pet3)
    {SE Added text}""
```

```
{SE Added text}""
```

```
{SE Added text}""
"Is there anything special you want to mention about ^pettype?"
: (Yes,Maybe, No) , dk, rf
TP5 (PT4)
    {SE Added text}""
```

```
{SE Added text}""
```

```
{SE Added text}""
"If so what you like to say?": open
```

BLAISE in Macedonia

Liljana Taseva, Mira Deleva, State Statistical Office of the Republic of Macedonia

1. Introduction

Starting from 1998 the BLAISE software is used at SSO for the need of Labor force survey. The survey is conducted in accordance to the Eurostat recommendations starting from 1996 and on annual level, while in the last four years it is conducted on quarterly level. The data gathering is done by the method of interview on printed questionnaires. Year by year, the basic content of the survey is extended and upgraded by adding of new questions and modules (for example, the module on long life learning, etc.).

The first survey was conducted in 1996 in the office and was realized on **MAINFRAME(UNISYS A12 MCP/AS** operative system. The data entry, controlling and performing the additional variables for tabulation and the tabulation itself were realized by using the principle of "BATCH PROCESSING". While data entering there was a limited "on-line" control for the accuracy of data entered. The errors were corrected on a listing and once corrected they were entered into the computer in index files.

This manner of work in the at that time information environment existing at SSO was acceptable, but with the fast development of the information technology and equipping of SSO with the same one, leded towards implementing a new manner of data processing in a client-server environment.

It was in 1998 when for the first time the application for LFS in **Blaise** was independently made by programmers in charge. In the preparation of the application, the Danish-Finnish Consortium that was present at that time at SSO gave the needed support.

In the first years, conduction of the survey was planned through engaging all the regional departments in the phase of data entry, aiming at reconsidering the advantage of this work manner as well as capabilities of SSO to realize the survey in a timely and quality manner.

The LFS data were entered by the persons from the regional departments trained for that job. In the application all nomenclatures and codes were included (municipalities, settlements, activities, occupations), so during the data entry except "on-line" controlling, coding of data was made. The whole work was organized in such a way that there was good cooperation with the regional departments and the methodological problems were solved on time. The data from the regional departments (Blaise files) were sent to SSO (in the beginning recorded on floppies and later with FTP) where data integration was made. Next step was data converting in SAS that is used for further processing (calculation of weights and tabulation).

By time, the program has been developed and advanced in direction to satisfy the users needs (subject matter department, team for data entry) aiming at providing quality data harmonized with European standards.

By equipping of SSO with IT equipment, at the regional departments PC of lower performances were installed so the data entry was transferred to SSO headquarters.

2. Short review of the functionalities of the application

The data entry starts by controlling of the identification data in already selected sample. Then starts the data entry for household (questionnaire A) and afterwards data entry for each person-member of the household (questionnaire B).

The application is started by double clicking of the icon for data entry and a screen is opened (picture 1)

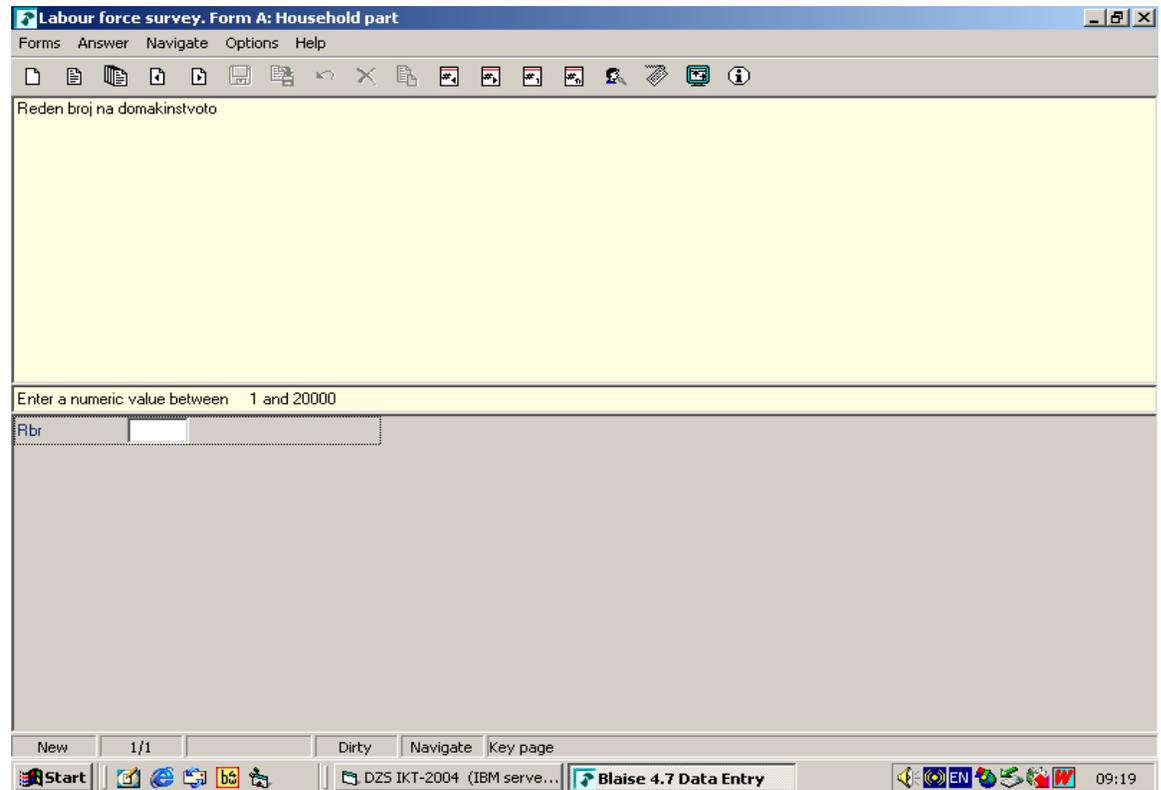
Picture 1



The interface is realized with Visual Basic 6.0

By entering a password and electing the button for data entry for households, persons, non-response, appears the data entry screen –

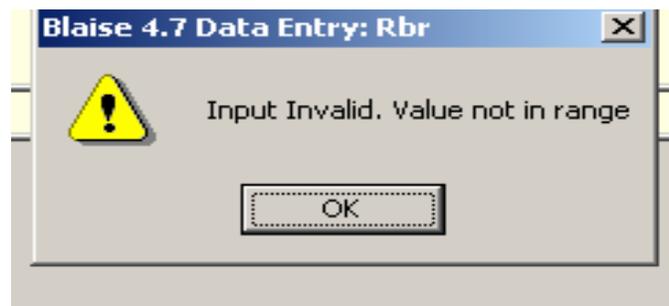
Picture 2



First, the identification of the household is entered, i.e. the ordinal number of the household in the field Rbr (ex. Rbr=15010). Each interviewer gets a list of households to be interviewed.

In case the interviewer entering the ordinal number of the household that is not encoded with the already selected sample (i.e. wrong ordinal number of the household), the program signals that that ordinal number does not exist in the sample (Input invalid. Value not in range) - Picture 3.

Picture 3



After checking the identification data in the sample, starts the data entry for responses for the household - questionnaire A .

The data entering is made in such a way that first household data are entered - questionnaire A - picture 4. Ten, individual data for the persons from the household - questionnaire B - picture 5. In case the household does not want to be interviewed, data on non-response are entered - questionnaire C - picture 6.

Picture 4

	PersonNo	Sex	BrDay	BrMth	BrYr	Age	MarStat	Soprug_a	Tatko	Majka	RelHH	Presens	Educlvl	Citispn	Trainon	Nacional	Over15
Person[1]	1					1											
Person[2]																	
Person[3]																	
Person[4]																	
Person[5]																	
Person[6]																	
Person[7]																	
Person[8]																	

Picture 5

	PersonNo	Den	Mesec	Godina	Sex	Q1	Q2	Q3	Q4	Q4a	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12
Person[1]	1	1	11	1970	1	1												
Person[2]																		
Person[3]																		
Person[4]																		
Person[5]																		
Person[6]																		
Person[7]																		
Person[8]																		

picture 6

Labour force survey. Form A: Household part

Forms Answer Navigate Options Help

Pricina za neizvršeno anketiranje

1. Adresata ne postoji na teren
 2. Prazen stan
 3. Stanot e demoliran
 4. Stanot se koristi za delovni nameni
 5. Domakinstvoto otkazuje da bide anketirano

6. Domakinstvoto e odsutno
 7. Nemoze da se kontaktira
 8. Starecko domakinstvo
 9. Drugo

Reason

Reject
Absent
TotNmo
MemAge
WorkNmo

New 12/12 Modified Dirty Navigate ARS

Start DZ5 IKT-2004 (IBM serve... Blaise 4.7 Data Entry 09:56

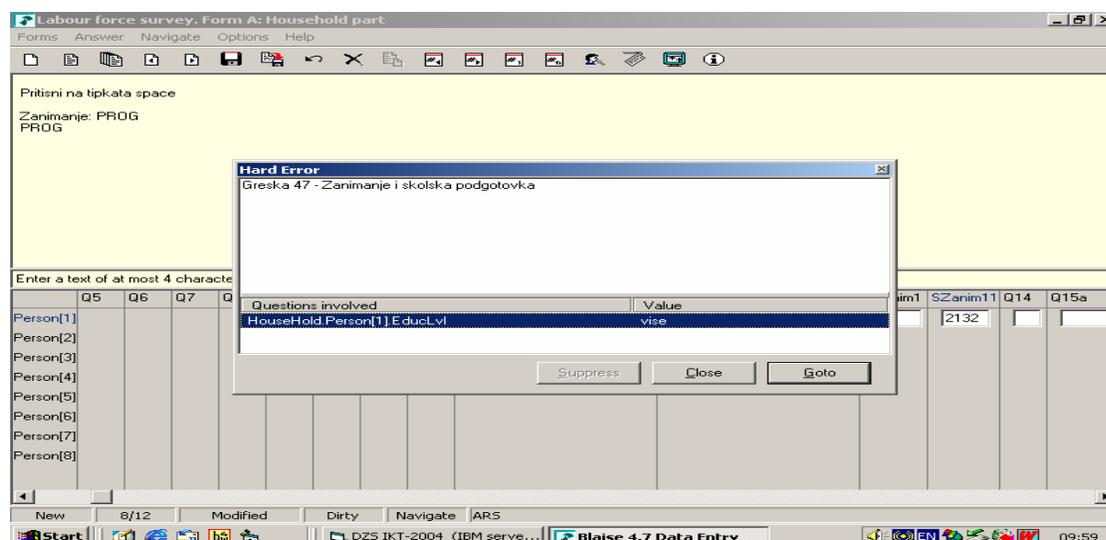
In the entering process all skips are such programmed that the person entering data just follows the filled in fields in the questionnaire and positioning of the right question is made by the program itself.

In the table of the questionnaire for households - questionnaire A, are entered all persons in the households while in the questionnaire for persons - questionnaire B are entered only those being of age 14 and above. Also there is no questionnaire B for permanent disabled persons.

The data that are entered are connected with on-line control (on the spot checking) so that illogical data can not be entered.

In case illogical data are entered an error dialogue appears that points out the error and fields included. In that case one goes to the field to be corrected and changes its content - Picture 7.

Picture 7



By clicking of **Close** and **Go to**, one goes to the fields that should be corrected.

In exceptional cases when we assumed that the answer is correct, we accept the error by pushing the button - of course if it is activated, i.e. the option **Suppress** is included in the program. Such accepted errors are registered and the questionnaires marked as erroneous are listed. From the delivered listing and when errors are defined a correction is made in data entry.

There is everyday data protection by copying them in several locations. This procedure is made each working day.

In the beginning the data were entered locally on each computer. Today the data and the programs are placed on a server (windows), and data being entered are recorded in one Blaise file on the server.

Within the framework of this investigation the State Statistical Office also conducted Test-Survey on households by using lap-top, but after the performed data processing it was decided to continue with collection of data with printed questionnaires.

In 2005 a test-application was made for entering of the survey material on the field by using lap-tops. The functionalities of the application remained the same.

The further data processing is in the SAS-software and finally the data are put on DB2-6000 basic data base.

From the experience so far before data tabulation and dissemination is made it is necessary to provide a documentation and data storage - the storage of Blaise files,

documentation from technical aspect: field, short description, name of variable, type, length, list of values.

Because we do not have the modules for direct access to DB2, data are transformed in SAS data files where further data processing is made.

The Labour Force Survey is the first BLAISE application in the Office. This application was for a long time one-of-the kind , and finally the advantages that this software provides were understood. So, today in the same software we have also other **investigation** . We expect their number to be higher and higher.

The expert support from the Slovenia in the field of Blaise and realized through SIDA in the recent years, has helped us a lot in organizing and advancing of the work with Blaise.

The Blaise in the Office is used for almost all surveys and lately is used for other research as well.

The strategy of SSO is Blaised to be used more for survey data entry (at the moment the way of data entry form questionnaires in printed form is used). In the future it is not excluded to apply other ways like interview with lap-top, telephone interview or via internet. Among other things it is necessary to establish and maintain the interviewer network meaning engaging skilled and quality interviewers.

From the so far implementation of Blaise we consider that it is exclusively efficient for the purpose it is used. The programmer staff fast and easy is adapted to its usage. It also enables simple, efficient and documented work.

Having in mind the experience up to date, as well as the wish for developing of new technologies, Blaise is one of the solutions for processing of the data from the Agricultural Census 2007.

Designing and Developing Person-based and Topic-Based Data Collection Instruments

Emily A. Johnson, Thomas C. Melaney, US Census Bureau

1. Summary

The American Community Survey (ACS) periodically conducts content testing in order to determine how proposed changes will affect the data collection operations. One of the recommended changes to the ACS was to modify the collection of demographic data from a person-based to a topic-based format. The ACS instrument used for Computer Assisted Telephone Interviewing (CATI) and Computer Assisted Personal Interviewing (CAPI) operations is programmed using Blaise 4.6.965. This paper examines the issues that were involved in developing the alternate ACS instruments that could be used for the collection of demographic data in person-based mode and in topic-based mode. The development issues included assuring that both of the data collection instruments conformed to the questionnaire design requirements and that both of the instruments conformed to the data processing requirements.

2. Background

The American Community Survey is a continuing survey that collects data from a sample of households throughout the United States. The data collection operation occurs each month with a different set of households. The U.S. Census Bureau collects and tabulates this data to produce current and reliable reports of the characteristics of households and household members. The ACS will replace the “long form” in future censuses and is a critical element in the Census Bureau’s reengineered 2010 Census plan. After extensive testing, the ACS reached full implementation in 2005. Currently, each month about 250,000 households receive a paper questionnaire by mail. After a month, about 100,000 households from which a paper questionnaire has not been returned are contacted by telephone to complete an interview. After another month, about 45,000 households for which the interview could not be completed by mail or telephone are visited for a personal interview.

Because of the large size of the data collection operation, and the high profile and the extensive use of the tables from the ACS, there is also an associated American Community Survey Content Test that is used to verify that proposed changes to the ACS will perform as expected without unintended consequences. In 2007, a modification was proposed which would be operationally tested in the ACS Content Test to change the data collection structure of the demographic section of the instrument. Until this time, the sections of the ACS program were in a person-based format; that is, all demographic information was collected about the first identified individual in the household, and then the next individual and so on. The proposed change was to collect the data for the demographic section in a topic-based format; that is, asking the first question in the section for each member of the household, then the next question for each member, and so on.

The question of whether the person-based or topic-based data collection method is more appropriate for certain survey topics have been tested at the U.S. Census Bureau. In 1997, a two-month test was conducted using alternate design structures of the ACS CATI instrument: person-based and topic-based. The results of this study were included in the paper “ACS/CATI Person-Based/Topic-Based Field Experiment” by Jeff Moore and Laurie Moyer of the Center for Survey Methods Research in the Statistical Research Division of the U.S. Census Bureau.

The test results examined length of interviews, response rates, refusal rates, and interviewer and respondent perceptions about the data collection process. The results generally indicated advantages for the topic-based approach. Some items were noted as needing additional research.

Following this test, the ACS retained a person-based approach pending additional review because the results for the questions collecting “income” data suggested that the person-based approach was more appropriate. In particular, the item non-response rates for the “wage/salary income” and for “total income” questions were somewhat greater using the topic-based approach.

For the 2010 Decennial Census operation, as well as the 2008 “Dress Rehearsal”, the U.S. Census Bureau decided to use a topic-based approach for the data collection operation. The questions for the Decennial Census ask about the demographic characteristics of persons within each household. When it is possible and appropriate, the ACS data collection question design should be consistent with that used for the decennial operation. Therefore, the ACS needed to be modified to use a topic-based approach for the data collection of the demographic questions for relationship, gender, date of birth and age, Hispanic origin and race.

The ACS 2007 Content Test needed to verify that moving to a topic-based design would not cause operational problems with the ACS 2008 data collection operation or with subsequent data processing operations. It was decided to use two instruments to allow a clear comparison between person-based and topic-based demographic data collection.

In the discussion that follows, please note that our example instrument is somewhat different from the actual Content Test instrument. These modifications were added to demonstrate appropriate enhancements that could have been incorporated had time permitted.

3. Approach to Design and Development

3.1 Client Request

The client was interested in conducting a test that would allow the comparison of the two data collection methods side by side. Two separate samples of respondents would be generated, with one group being asked questions in the original person-based method, and the other in the topic-based method. This required the development of two separate instruments, one using the person-based approach currently used in production for the American Community Survey, the other using the new topic-based approach. Other than differing demographic sections, the two instruments had to be kept identical. Output processing staff required that the same data be output from both instruments.

Despite the two demographic sections operating differently, the sponsor’s requirement was that the two instruments have the same look and feel, with an identical table layout, and all of the person data still being displayed across a row. With the new topic-based instrument closely resembling the instrument being fielded in production, a minimal amount of training would be required. It would also lower the impact on the interviewer of changing from one content test instrument to the other.

A free-form navigation approach to collecting the data in the table would not fulfill the client’s requirements, as it was not sufficiently structured. The goal was to continue to guide the interviewer from question to question; however, the client preferred to move through the table column by column, instead of row by row.

3.2 Specification Development

The two programs had a similar structure, with many modules outside of the demographic section being identical, and the naming conventions between the instruments remaining nearly the same.

The specification needed to indicate the new question flow through the topic-based demographic section. Because the order in which the questions were to be asked was not changing, the spec-writer was able to start with the current person-based specification and modify it. In a “Special Instructions” section of the field level specification, the spec-writer documented that a question was to be asked of each person in the household before moving on to the next question (see Figure 3.2.1).

Figure 3.2.1 – Specification example describing flow through grid

<p><u>Special Instr.</u></p> <p>2. Interviewers are to navigate vertically in the grid, that is, each question set must be answered for each person listed before moving to the next question set for each person in the grid. The relationship items for each respondent (REL P) will be answered for each person before moving to SEX.</p>

Certain questions were to be answered as groups, and the specification needed to account for these exceptions to the straight topic-based approach. For example, the Hispanic Origin questions (HISA, HISB, and HISW) had to be answered as a question set, asking the leading questions (HISA), then the probe questions (HISB and HISW) if appropriate, for a single person before moving on to the next household member. Again making use of the “Special Instructions” for a specific field, the sponsor was able instruct the programmer as to whether or not the field was to be part of one of these special topic groups (see Figure 3.2.2).

Figure 3.2.2 – Specification example describing flow through grid with question set

<p><u>Special Instr.</u></p> <p>2. Interviewers are to navigate vertically in the grid, that is, each question set must be answered for each person listed before moving to the next question set for each person in the grid. The Hispanic origin items for each respondent (HISA and HISB and HISW, if applicable) will be answered for each person before moving to the race questions.</p>

A significant reason for asking the questions in a topic-based approach was to make use of different wording that would ease respondent burden and improve interview flow. By

asking the same question of each person, the full question text would not need to be repeated, and an abbreviated form of the question could be asked for additional members in the household. Some questions could have three or four different variations, depending on how many times the question has already been asked and what question preceded it if it is part of a topic group with probe questions. Again, the specification needed to account for these significant wording changes. The specification writer created fills in the question text and based the fill text on whether the question had already been asked of a preceding household member.

In the question wording specification example below (see Figure 3.2.3), one of the more complex scenarios in the demographic section, there is question text including an introduction and example list to be read the very first time the RACE question is asked of someone in the household, along with slightly shorter text for the second time the question is asked. There follow two wording possibilities for the third or more times the question is asked – one if the previous person was asked any follow-up race questions and one if only the initial question was asked.

Figure 3.2.3 – Specification Example with Variable Wording

<u>Question Text</u>	
	If first time this question is asked:
	I'm going to read a list of race categories. You may choose one or more races. {Fill 1: For this survey, Hispanic origins are not races.}
	{Fill 2: Is <Name>/ Are you} White; Black, African American, or Negro; American Indian or Alaska Native; Asian; Native Hawaiian or Other Pacific Islander; or Some other race?
	<11> White
	<12> Black, African American, or Negro
	<13> American Indian or Alaska Native
	<14> Asian
	<15> Native Hawaiian or Other Pacific Islander
	<16> Some other race
	If second time question is asked:
	{Fill 2: Is <Name>/ Are you} White; Black, African American, or Negro; American Indian or Alaska Native; Asian; Native Hawaiian or Other Pacific Islander; or Some other race?
	<11> White
	<12> Black, African American, or Negro
	<13> American Indian or Alaska Native
	<14> Asian
	<15> Native Hawaiian or Other Pacific Islander
	<16> Some other race
	If not first or second time question is asked and previous person RACT=13, 14, 15 or 16:
	What is {Fill 3: <Name>'s/your} race?
	{Fill 2: Is <Name>/ Are you} White; Black, African American, or Negro; American Indian or Alaska Native; Asian; Native Hawaiian or Other Pacific Islander; or Some other race?}
	<11> White
	<12> Black, African American, or Negro
	<13> American Indian or Alaska Native
	<14> Asian
	<15> Native Hawaiian or Other Pacific Islander
	<16> Some other race

Figure 3.2.3 – Specification Example with Variable Wording (continued)

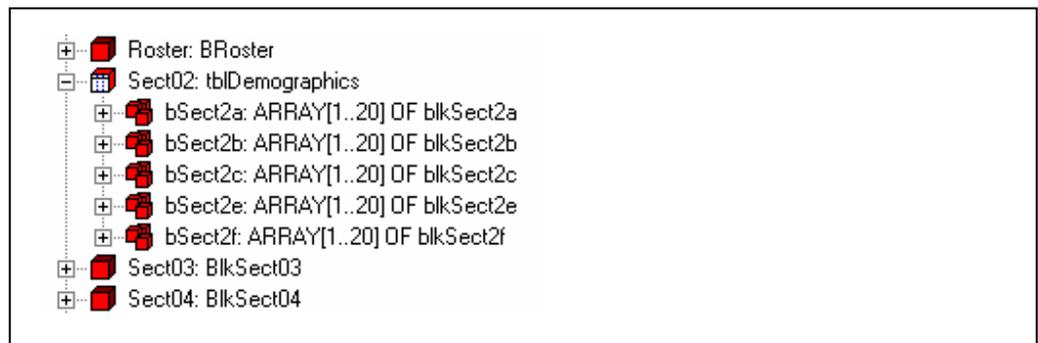
Question Text	<p>If not first or second time question is asked and previous person RACT NE 13, 14, 15 or 16:</p> <p>How about {Fill 4: <Name>/you?} ({Fill 2: Is <Name>/ Are you} White; Black, African American, or Negro; American Indian or Alaska Native; Asian; Native Hawaiian or Other Pacific Islander; or Some other race?)</p> <p style="padding-left: 40px;"> <11> White <12> Black, African American, or Negro <13> American Indian or Alaska Native <14> Asian <15> Native Hawaiian or Other Pacific Islander <16> Some other race </p>
----------------------	---

3.3 Design Considerations and Implementation

The Content Test instrument would be developed using Blaise 4.6 build 965, to match the version with which the current 2008 ACS Production instrument was being developed.

In order to move through the table in a columnar, or topic-based, fashion, it was necessary to break up each question into its own table. The demographic block itself was defined as a table, with each question an arrayed block inside – two relationship question blocks (Sect2a1 and Sect2a2), a gender block (Sect2b), an age block (Sect2c), two Hispanic origin blocks (Sect2e1 and Sect2e2), and a race block (Sect2f). Only the gender block is purely topic based. It didn't make sense to break up the three fields that collected the month, day, and year, nor the age probe questions that followed them. Neither was it logical to move the “specify other” race questions apart from the leading race question. Several of the other question blocks also contained separate questions depending on whether a telephone or personal interview was being conducted. Eventually, the survey methodologists decided that they preferred to also collect the relationship questions and Hispanic origin questions as groups instead of individually, leaving us with five tables instead of the initial seven (see Figure 3.3.1).

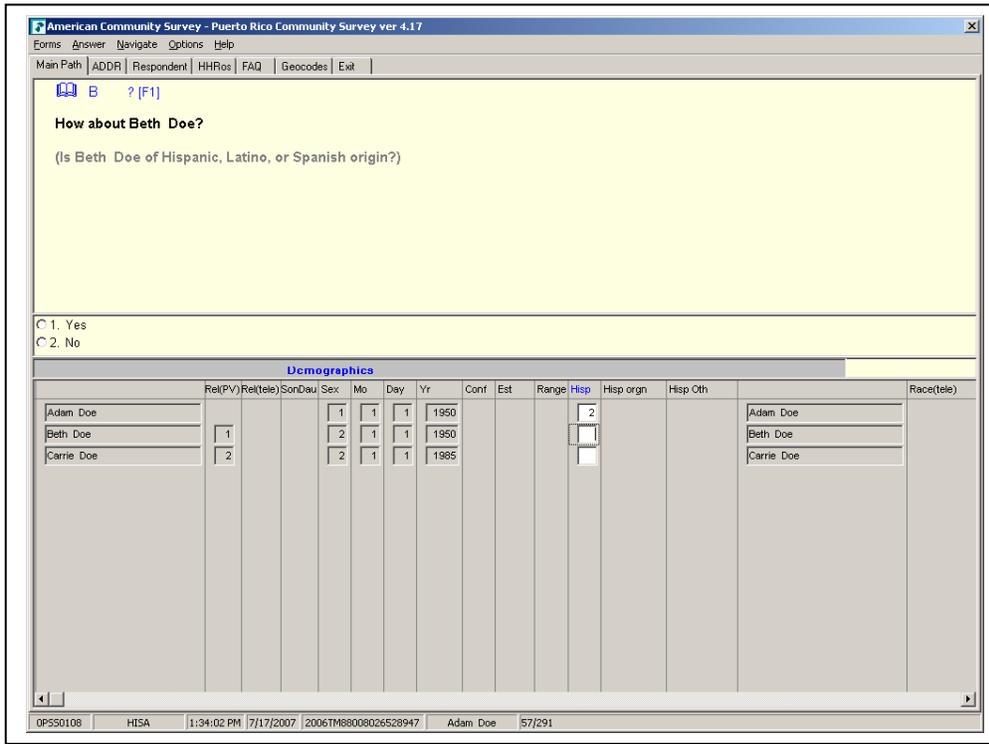
Figure 3.3.1 – Data structure of the Demographic Section



The original person-based instrument contained all of the data in a single table. By breaking the topic-based instrument into pieces, the entirety of the demographic data was no longer available to the interviewer at any point during the demographic portion of the interview. To circumvent this problem, the programmers decided to display, or “show”, all of the data that was not currently being “asked”. In this way, the interviewer would

still progress down a single column without losing the ability to see all the data that had been collected thus far (see Figure 3.3.2).

Figure 3.3.2 – Screenshot of Table With All Data Displayed



Nonetheless, a straight “show” of the extra data was not ideal, as not all of the data would always be on-path to be collected and this would cause empty fields to be “shown”. This meant that rules were required even for showing the fields not collected in the current table. Rather than painstakingly coding the rules to “show” each individual field, the programmer placed the fields into blocks that mimicked the sub-blocks that were being “asked”, and performed the “show” method on the entire block, and thus all of the fields it contained (see Figure 3.3.3).

Figure 3.3.3 – Code from the Sect2a block demonstrating ASK vs. SHOW of sub-blocks

```

bRel_Questions      ({IMPORT}  IN_lang_ext, LNO, IN_NAME, bSex_Display.SEX,
                    IN_HHolderVar, IN_HHNAME, IN_RESPLINE,
                    IN_MODE, IN_TQA_FLAG, IN_PV_TELE,
                    {TRANSIT} TR_REL, TR_RelNum)

bSex_Display.SHOW  ({IMPORT}  IN_lang_ext, LNO, IN_NAME, IN_RESPLINE,
                    {TRANSIT} TR_SexNum)

bAge_Display.SHOW  ({IMPORT}  IN_lang_ext, LNO, IN_NAME, bSex_Display.SEX,
                    IN_HHolderVar, IN_HHNAME, IN_RESPLINE,
                    {TRANSIT} TR_TempDOB)

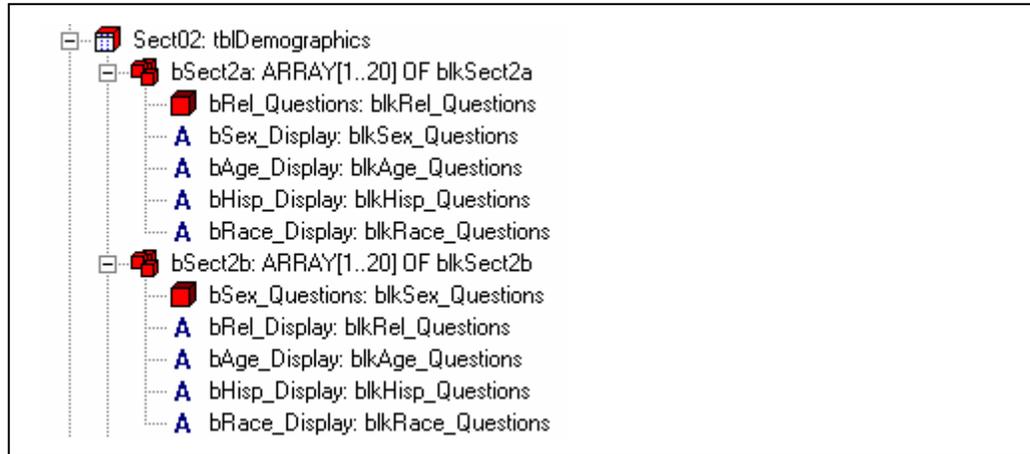
bHisp_Display.SHOW ({IMPORT}  IN_lang_ext, LNO, IN_NAME, bSex_Display.SEX,
                    IN_RESPLINE, IN_MODE, IN_TQA_FLAG, IN_PV_TELE,
                    {TRANSIT} TR_HISAFlag, TR_HISANum, TR_HISBNum,
                    TR_RaceIntroFill, TR_RaceIntroFillSP)

bRace_Display.SHOW ({IMPORT}  IN_lang_ext, LNO, IN_NAME, bSex_Display.SEX,
                    IN_RESPLINE, IN_MODE, IN_TQA_FLAG, IN_PV_TELE,
                    TR_RaceIntroFill, TR_RaceIntroFillSP,
                    {TRANSIT} TR_RACNum, TR_RACFlag, TR_RCW1Num, TR_RCWAGNum,
                    TR_RCWPGNum)

```

With each table being basically five copies of the same table, but with a separate set of questions being “asked” versus “shown” in each, the programmer was going to have to maintain five sets of the same questions. To cut down on the amount of duplicated code, a block type was created for each question set that would be reused in each of the five tables, both those question blocks being “shown” and those being “asked” (see Figure 3.3.4). Because the block was defined in a single place, any changes made to it would apply to all tables. This improved efficiency, as well as long-term maintainability.

Figure 3.3.4 – Sub-blocks defined as fields or auxfields, of same block type



Another benefit of using the same blocks in each table was the ability to copy an entire block at once at the higher demographic level, making it very easy to keep all five tables up-to-date with the current data without having to pass large numbers of parameters from one to another and compromise performance (see Figure 3.3.5).

Figure 3.3.5 – Code from the Sect2a block demonstrating ASK vs. SHOW of sub-blocks

```
bSect2a[I].bSex_Display := bSect2b[I].bSex_Questions
bSect2a[I].bAge_Display := bSect2c[I].bAge_Questions
bSect2a[I].bHisp_Display := bSect2e[I].bHisp_Questions
bSect2a[I].bRace_Display := bSect2f[I].bRace_Questions
```

We found ourselves dealing with a demographic section five times the size of the original demographic section. To cut down on the size of the database, and minimize confusion with multiple copies of fields being output from the instrument, all sub-blocks that were being “shown” were defined as auxfields (see Figure 3.3.4). Therefore, in each table, only the sub-block that was being “asked” would actually be output. Also, the auxfields are not saved in the database, cutting down on the amount of time needed to save to the database upon closing the instrument.

The change to the demographic section had an impact on other parts of the instrument. Demographic data gets used throughout, for generating fills and determining question pathing. With the table now being split into five distinct tables, the full paths of fields changed. To reduce how many places the field name would need to be modified, explicit parameters were used as extensively as possible.

There were several obstacles that had to be overcome by using this particular design. Each block was being “asked” once, but “shown” an additional four times. When the

“show” method was applied, certain checks would be triggered again, meaning a single check might appear a total of five times. In some cases, where the check was conditioning on a calculated value, as opposed to a field within the block, none of the involved fields would be on path, and the instrument would get stuck. To circumvent this problem, a flag was added so that the block would know whether or not the “show” or “ask” method was being applied, and in the case of the “show”, the check would be ignored. In another situation, several checks had been embedded within the sub-blocks, but conditioned on data from several sub-blocks. By moving these checks up to the main demographic level, not only was the problem of the checks being triggered multiple times avoided, but also a number of generated and explicit parameters were eliminated.

3.4 Comparison to Topic-based Demographics in SIPP

In a previous project, the Survey of Income and Program Participation (SIPP), the authoring staff was required to program the demographics section of a Blaise instrument using a topic-based approach. Unlike ACS, however, the SIPP survey designers did not request to display the demographic data within a single table. Instead, each arrayed block was displayed separately. The existing SIPP instrument was programmed using the Computer Assisted Survey Execution System (CASES), and having only ever worked with SIPP as designed in CASES, the interviewers were accustomed to seeing only one field at a time. ACS, on the other hand, was already coded as a Blaise instrument, and had been fielded in production with the person-based mode, with all fields in one table, for several years. Therefore, to smooth the transition from person-based to topic-based, it was necessary to make the display change virtually transparent.

4. Conclusion and Recommendations

4.1 Flexibility Within the Table

While the ability to use free-navigation within tables in 4.7 has greatly increased the flexibility of data collection in Blaise instruments, it would be helpful to have the ability to design an instrument selecting the direction of flow through a table without sacrificing a structured interview where the interviewer continues to be guided from question to question.

In this example, where we had combined vertical and horizontal movement, it would probably still be necessary to create multiple arrayed blocks. All five arrayed sub-blocks were included in the same table. However, each sub-block was displayed on its own page, making it necessary to create copies to mimic a single table. The task would have been greatly simplified had Blaise had the ability to display all five arrayed sub-blocks on the same page. An additional drawback of the current implementation was that the interviewer could not use the mouse to skip backwards or forwards to edit a field only being displayed on the current page. It was necessary to use the arrow keys or page up and page down keys to navigate. While CAPI field interviewers are primarily accustomed to using the keyboard to navigate, should an interviewer choose to use the mouse, the fields being “shown” could cause confusion.

The way in which Blaise performs the “show” method on a block caused problems in this example. Checks were triggered even in the case where all fields involved in the check were themselves being “shown” instead of “asked”. It would be easier if checks within the block were treated the same way they were treated during a “keep” method.

4.2 Output Processing

For this particular project, the Blaise database was converted to a flat ASCII file for output processing. If the data were being output in an ASCII relational format, it is important to note that the five arrayed blocks would be output in separate files, and it would be helpful to have linking variables, such as the person line number, added to each sub-block to aid in piecing together a person's data.

5. Acknowledgements

5.1 Acknowledgements

Many individuals have worked and continue to work on components of the American Community Survey to ensure its successful operation. The authors would like to include grateful acknowledgement to all of our colleagues from the U.S. Census Bureau whose work directly contributed to the instrument development discussed in this paper. We thank in particular Li Li, Kenneth Stulik, Gianna Dusch, David Raglin, Courtney Reiser, Diane Cronkite, Todd Hughes, and Debra Klein for their work in the coordination, design, programming and testing of the instrument used in this phase of the project.

This document is to inform interested parties of ongoing research and to encourage discussion of Blaise instrument design issues. The views expressed are those of the authors and not necessarily those of the U.S. Census Bureau.

5.2 References

Moore, J. and Moyer, L. (2002), ACS/CATI Person-Based/Topic-Based Field Experiment, U.S. Census Bureau, Statistical Research Division, Research Report Series (Survey Methodology #2002-04), issued February 19, 2002

Coping with people who just won't stay put: The Use of Blaise in Longitudinal Panel Surveys

Colin Setchfield, Office for National Statistics, UK

1. Introduction

The UK's Office for National Statistics (ONS) has conducted the General Household Survey (GHS) since the survey's inception in 1971. It is a multi-purpose continuous national survey of people living in private households, comprising a household section (where all questions are addressed to a household reference person) and individual sections (where questions are asked individually of all resident adults in the household aged 16 and over).

In 2005, in order to meet new requirements for producing comparable data for the European Union's Survey of Income and Living Conditions (EU-SILC), these were incorporated into GHS¹. The result was the implementation of a rotating four-year panel design, with first follow-up interviews in 2006. The sample would be interviewed in each of the three subsequent years after it was first included; each of the four years is referred to as a 'Wave'. (In order to replenish the sample, a quarter of the sample is replaced each year.)

Other panel surveys conducted by ONS – such as its Labour Force Survey (LFS) – always returned to the original sampled addresses at each subsequent wave, and interviewed the residents there: irrespective of whether they were the interviewees at the previous occasion. Contrariwise, those interviewed at Wave 1 on GHS were defined as original sampled members. It was them, and not necessarily the address at Wave 1, in which successive waves would be interested.

These original sample members (OSMs - the household members at Wave 1) remain part of the sample for the remaining waves, unless they die, move into an institution or move abroad. If they move elsewhere within Great Britain, and are resident there at a later wave, they are interviewed at their new address.

At successive waves, any other people who move in with the original sampled members from Wave 1 (irrespective of address) also become eligible for interview. These later additions only remain eligible so long as they reside at an address where at least one original sampled member from Wave 1 also resides.

A method of feeding data forward across to the next wave had been previously established at ONS, and with minor adaptation this was applied to the GHS to become the standard ONS approach for rotation². It was, however, this tracking of people's movements and the subdividing of households which had split that became the main challenge in developing the Blaise datamodel.

¹ The merged survey now uses the acronym GSL though in this paper the more familiar GHS has been retained throughout.

² "Rotation" and "feeding forward" are interchangeable terms with ONS. They both refer to the transfer of data collected at one wave into the next subsequent wave's datamodel.

This paper details both the ONS method of rotating data, and also the filtering method to change the household array across waves: used in order to cater for the removal and addition of household members and forcing removed members into new household cases for interviewing or processing. Its conclusion will review the impact of such complicated code, particularly in the light of organisational changes experienced within ONS.

2. Feeding forward data: establishing a standard ONS approach

2.1 Background

Feeding data forward from one wave to another was already used within ONS on its Labour Force Survey (LFS) prior to the redevelopment of the GHS. The LFS method was relatively straightforward. The data from the previous wave were read into the individual cases when created, prior to their release into the Field. Consequently, they were automatically populated in the Fieldpane within the interview grid even before interviewing began. This method predated ONS's use of Manipula, and was dependent on previously-used Clipper programs. It was therefore not a basis for feeding data forward on GHS.

2.2 Use of External database

A method for rotation using Blaise, was introduced by Tim Burrell for the introduction of the Annual Population Survey (APS) in 2004, having previously been used on a number of ad hoc surveys. Rather than pre-filling the questionnaire with data from the previous wave; the data were provided as an External database, allowing rotation to be processed within the datamodel itself.

All data that needed to be fed forward to the following wave were to be placed in a single file. Feeling that, in some situations, there may be capacity issues if the resulting file proved too large, it was felt that it would, in these instances, be appropriate to subdivide the file and use more than one External file. On the APS, data were divided into two files: one holding household-level information and another holding person-level information, though this was mainly to meet the processing needs of the LFS.

On GHS a single External file was provided combining both household-, and person-, level information for the whole sample. Nevertheless, in order to optimise the External file, only those fields whose data were to be fed forward were included.

Originally, consideration had been given to an option to “pick and mix” individuals from different addresses, should someone move from one sampled address to another: hence every record in the sample was provided within the External file. Writing Blaise code to facilitate this merging of data was deferred, to allow time to investigate frequencies of this happening. With no such cases observed, this option was dropped, requiring interviewers simply to re-enter the data of a sampled person who should happen to move in with other sampled members. As a result, the External database needs only to include the one related record from the sample rather than the full record. Though this change has yet to be implemented on the GHS, a single-record External file is being used for ONS's next new panel – the Household Asset Survey.

On GHS, with waves a year apart from each other, a separate Keeping-in-Touch Exercise (KITE) was provided between waves in order to update contact information when

applicable. One option would have been to use the KITE data as a second External file at the subsequent wave in order to supplement or update the data from the previous wave. Instead it was decided that GHS data would be fed forward into the KITE, so that at the subsequent wave only one External (the KITE database) was required, thereby reducing the number of dependencies.

In preference to reading the External file at each block in the questionnaire, a separate holding block (QRotate) was provided, which read the External database of the previous data and to which the data were copied across. The External file was read only once; from this holding block, the fed-forward data could be manipulated and assigned to other blocks within the questionnaire, without further reference to the External.

2.3 Methods of feeding forward data

Three methods of feeding forward data were available.

- Automatic feeding of data into Fieldpane
- Showing fed-forward data in Infopane as an interviewer instruction
- Showing fed-forward data in Infopane as part of the read-out text of the question

Dependent interviewing – using previously-collected data either to remind the respondent of his or her previous answers or to probe for changes since then – is widely used in panel surveys. It helps to reduce respondent burden, as well as having beneficial effects on interview length, and reports have shown evidence for improved data quality.

Automatic feeding of data into Fieldpane

This was the method that had been utilised on the LFS. As that survey grew over the years, this method helped to contain an exponential growth in questionnaire length at subsequent waves. Such was the extent that this continues to be the method of choice for the LFS even when it becomes part of the Integrated Household Survey in 2008.

It was felt, however, that requiring interviewers to re-enter answers was more methodologically sound (Jäckle (2006) – see below). Therefore, apart from at a few fields in GHS, this method is restricted in ONS to LFS alone and is not seen as the standard approach.

One of the dangers of placing data from the previous wave in the Fieldpane was interviewers unintentionally passing over whole sections of questions, simply by accidentally pressing one of the navigation buttons on the laptop (such as <Page Down>).

As mentioned above, this method was employed in a few instances on the GHS. These were (1) where data were unlikely to have changed apart from in the most exceptional circumstances, or (2) where a filtering question could be used to check whether the previous situation had changed. This first category comprised the questions on sex, date of birth, and also household accommodation, though allowing an overwrite option in case the details had changed or had been recorded incorrectly before. The second category comprised questions on occupation and qualifications. For these, a filter question was introduced at subsequent waves, because of the level of information associated with them.

The filter question provided the full information on occupation or qualifications held, asking whether the person held exactly the same job as before or whether she had obtained any additional qualifications since last wave. Depending on the respondents' answer at the filter question, the ensuing questions on the topic were either automatically filled and passed over or the interviewer was routed back through them.

Figure 1: Method and Blaise Code for feeding data into Fieldpane

Method:

In the FIELD section,

- add the Empty attribute to the relevant field(s) (to allow field to be shown as blank on deleting previous data before entering new data)

In the RULES section,

- name the field
- add a condition to apply only if that Field is empty
- (for Fields related to individuals) add a secondary condition to apply only if the person is not a new household member
- add a final condition to apply only if data exists within the holding block (QROTATE)
- instruct Blaise to assign the value of the fed-forward data into the live Field

Example Code:

Sex

```
IF Sex <> RESPONSE THEN
  IF QNames.QBNames[j].AxNewPer=No THEN
    IF QRotate.QBRotate[j].RSex =RESPONSE THEN
      Sex:=QRotate.QBRotate[j].RSex
    ENDIF
  ENDIF
ENDIF
```

Showing fed-forward data in Infopane (as question text or instruction)

There were concerns that overuse of the option of automatically feeding data into the Fieldpane could lead to a deterioration in data quality. The fear was that interviewers could become too reliant on it and not check sufficiently whether the information had changed since last call.

Consideration was given to using variable text fills within the question wording, in order to reference the answer from the previous wave before asking the question itself. For example, the question, "How many bedrooms do you have?" at a subsequent wave would read, "Last time, you said you had four bedrooms; is this (still) correct?"

Within ONS, variable text fills are used sparingly. For us, it has always been important to keep Blaise code easily read and intelligible; an excessive use of text fills has been seen as counter to this. Similarly, there is a concern that an over-reliance on text fills could impact on the focus and professionalism of interviewers, encouraging them to read the questions without considering the context.

It was also felt that showing the fed-forward data as an interviewer instruction, and not read into the spoken text of the question, safeguarded against the respondent being unduly influenced by the answer that they gave at the previous wave.

Figure 2: Method and Blaise Code for displaying data in Infopane

Method:

In the AUXFIELD section,

- create an auxfield to display the variable text into which the previous data will be fed forward (it may also be necessary to create locals - LRData - for fed-forward data set up as enumerated Sets). For GHS, a single Auxfield was defined (AxRData) and was set back to empty after each use

In the RULES section,

- first set the Auxfield to blank (e.g. AxRData:=")
- if fed-forward data is set up as an enumerated Set and the Field name is not clear enough to be used within the variable text, define the text to be used for each option by using the Locals you have created
- (for Fields related to individuals) add a condition to apply only if the person is not a new household member
- add a secondary condition to apply only if data exists within the holding block (QROTATE)
- define the text to be displayed using the Auxfield (in combination with any Local if necessary)
- the toggle @R was been defined for use with instructions displaying fed-forward data, this formats the text in a dark-orange red.

Example Code:

```
LRData:=""
AxRData:=""
IF DVAge >=16 THEN
  IF QRotate.QBRotate[i].RMarStat = NevMarr THEN
    LRData := 'Single - Never Married <1>'
  ELSEIF QRotate.QBRotate[i].RMarStat = MarrLiv THEN
    LRData := 'Married and living with spouse <2>'
  ELSEIF QRotate.QBRotate[i].RMarStat = Separated THEN
    LRData := 'Married but separated from spouse <3>'
  ELSEIF QRotate.QBRotate[i].RMarStat = Divorced THEN
    LRData := 'Divorced <4>'
  ELSEIF QRotate.QBRotate[i].RMarStat = Widowed THEN
    LRData := 'Widowed <5>'
  ENDIF
  IF QNames.QBNames[i].AxNewPer=No THEN
    IF QRotate.QBRotate[i].RMarStat = RESPONSE THEN
      AxRData:='@RLast time Marital Status was recorded as ' +
      LRData +'@R'
    ENDIF
  ELSE
    AxRData:=""
  ENDIF
MarStat
```

Jäckle (2006) warns against ‘proactive dependent interviewing’ – reminding respondents of previous answers – “because of the potential under-reporting of change over time”.

Consequently, GHS adopted the alternative approach of ‘reactive dependent interviewing’ – using previous answers only to check information received. Data from the previous wave were displayed in the Infopane as an interviewer instruction only (not read out to the respondent): leaving the question text unaltered. This required the interviewer to re-enter the data each time. Though this would take more time to interview at the subsequent wave, it was felt that data quality was safeguarded.

3. The Filter Block: Disposing of and reallocating unwanted people

3.1 Background

The new Rotation method for feeding data forward across waves only addressed part of the longitudinal aspect of GHS. It was most suited to panel surveys where the respondents never moved from their original address or, if they did, they dropped from the sample.

GHS was different. It allowed for people to become eligible at later waves even though they had not been interviewed in previous waves. It allowed for people to retain their eligibility even if they moved address. It allowed for a single address at Wave 1 to spawn eligible households at multiple addresses across the country as people moved out of their original home into new homes with or without those they had previously lived with, and also with or without others with whom they had never previously lived.

To cope with these requirements, a Filter block (QTFilter) was developed in order that, before interviewing at the household, the current eligibility status of each member at the previous Wave could be recorded.

The Filter block, however, grew in function, also serving to (1) incorporate data on new contact details for ‘Movers’, (2) act as a controlling block for all multi-households created at subsequent waves, and (3) facilitate checks so that a case cannot be transmitted without all necessary additional households being opened.

3.2 Structure of Filter Block

In terms of its Fields, the structure of the Filter Block was relatively straightforward. For each person recorded at the previous interview, the following details were collected.

- Current Status, with categories defined as:
 - Resident here – eligible to be interviewed
 - Resident here – under the age of 16
 - Moved from Household 1 – Now resident locally; details known; can interview
 - Moved from Household 1 – Now resident elsewhere in Great Britain; details known; reallocate
 - Moved from Household 1 – Now resident at unknown address
 - Ineligible – Died since last call
 - Ineligible – Now in institution (for 6 months or more)

- Ineligible – Now resident abroad (for 6 months or more)
- Ineligible – Mover at GSK, new case already created; or, No OSM left (*GSK is the KITE questionnaire for GHS*)
- A Derived Variable Field recording whether the person was an OSM
- For those who have moved, two Fields: one asking whether the person moved to the same address as another person already coded, and the other to record the date a person moved
- For these movers, Fields are provided to record details of new address, telephone number and email, or these if details are not known a Field to record what action was taken by the interviewer in order
- For those who have become ineligible, two Fields: one to record the country to where a person has moved, and the other to record a date of death or committal to an institution (both only when appropriate)

The complicatedness of the block, however, lay within its routeing, particularly in terms of how it would be used as a “gate-keeper” to determine which individuals from the previous interview should continue to be interviewed using the same case and which should be assigned to new cases in order to interview them elsewhere or to code them off out of the sample.

3.3 Different category – different case

ONS interviewers and its Casebook system (for managing cases in the Field) were both used to dealing with split households. A case could be spawned in the Field to create a second, third or fourth version of it. It would retain the same Primary key elements of Area number and Address number, and would increment the remaining element – Household number – by one for each new case created. In usual circumstances, this was used to cope with multihousehold addresses; at subsequent waves, however, GHS was not usual.

After its first wave, GHS’s sample was relatively fixed. Only OSMs of households at Wave 1 (and people they live with at the time of re-interview) would be eligible for interview. As such, no new multihousehold address detected at subsequent waves would be eligible, unless it comprised of at least one of the original household members.

The option of creating additional households at addresses was therefore used instead as a way of processing the people from last interview into their separate common categories. All people who continued to be resident at the same address would be included all together in one household (if there were any in this category, they would continue to be interviewed at Household 1). Those who had moved would be designated the next available household on the following basis.

1. Those who had moved locally within the same sampled area and therefore still able to be interviewed by the same interviewer. If more than one person moved into the same address, they would be interviewed as one household; if a person moved without any other OSM, they would be designated their own individual household.

2. Those who had moved outside of the sampled area, and therefore it was more effective to reallocate to another interviewer working closer to the new address. Again, those who moved together would be designated the same household.
3. Those who had moved to an unknown address.

Finally, all those who were no longer eligible for interview (those who had died, been institutionalised, or who moved abroad) would be designated a single household in order to be coded out of the sample. The same Final Outcome code applies to all household members who become ineligible and therefore separate households were not required for these.

Having categorised the previous residents into groups for interviewing or coding off, it was next necessary to determine whether any member of the group was an original household member. If none was, that group would no longer be eligible for interview.

Figure 3: Method and Blaise Code for designating individuals to separate households

Method:

In the FIELD section,

- create a field to record whether person is an OSM or living with an OSM
- create a field to record household number at which person should be processed
- create counters for number of Local Movers, Distant Movers, and Movers to unknown addresses,

In the RULES section,

- first field is simply a matter of assigning a value on the basis of the OSM field of the person or, if a mover, of the person with whom they moved
- compare Current Status field of person with the field of (if a mover) whether they moved with someone else and with the field above
- if person still eligible and resident at same address allocated to household 1
- set each mover counter to - 1
- count number of movers by category so long as they moved locally without anyone else
- on counting person, assign household on the basis it will be the sum of 1 plus the aggregate of all counters processed to that point in order

Example Code:

LGB:=(-1)

{Local for counting the number of distant movers}

LLocal:=(-1)

{Local for counting the number of locals movers}

LDK:=(-1)

{Local for counting the number of movers to unknown addresses}

FOR i:=1 TO 16 **DO**

IF (QBFilter[i].CurStat=Resident **OR** QBFilter[i].CurStat=UnderAge) **THEN**
QBFilter[i].MemHH:=1

{Those still resident are defaulted to Household 1}

ELSEIF (QBFilter[i].CurStat=MovedInt **AND**

(QBFilter[i].MovedWth=EMPTY **OR**

QBFilter[i].MovedWth=17)) **THEN**

LLocal:=LLocal+1

QBFilter[i].MemHH:=1+LLocal

IF NResident=1 **THEN**

QBFilter[i].MemHH:=QBFilter[i].MemHH+1

ENDIF

Example Code (continued):

```
ELSEIF (QBFilter[i].CurStat=MovedGB AND
(QBFilter[i].MovedWth=EMPTY OR
QBFilter[i].MovedWth=17)) THEN
  LGB:=LGB+1
  QBFilter[i].MemHH:=1+LGB
  IF NResident=1 THEN
    QBFilter[i].MemHH:=QBFilter[i].MemHH+1
  ENDIF
  IF NMovedInt>0 THEN
    QBFilter[i].MemHH:=QBFilter[i].MemHH+NMovedInt
  ENDIF
ELSEIF (QBFilter[i].CurStat=MovedDK AND
(QBFilter[i].MovedWth=EMPTY OR
QBFilter[i].MovedWth=17)) THEN
  LDK:=LDK+1
  QBFilter[i].MemHH:=1+LDK
  IF NResident=1 THEN
    QBFilter[i].MemHH:=QBFilter[i].MemHH+1
  ENDIF
  IF NMovedInt>0 THEN
    QBFilter[i].MemHH:=QBFilter[i].MemHH+NMovedInt
  ENDIF
  IF NMovedGB>0 THEN
    QBFilter[i].MemHH:=QBFilter[i].MemHH+NMovedGB
  ENDIF
```

3.4 Concertinaing the household per case

The first part of the Filtering process had thus provided each household member from the previous interview with their current eligibility status and also designated a grouping or 'household' in which to process or interview that individual.

The second part of the Filtering process was then the feeding forward of data from last interview (discussed above), according to the eligibility status and the designated household. In addition to this, a block was required to list names of those persons who had joined the household since last interview. It was necessary for this filtering and rotating of data to be undertaken within the Blaise code prior to the questionnaire proper, in order that household arrays throughout the questionnaire were informed by the new household composition.

As detailed above, feeding forward data from one wave to another can be simply achieved by assigning data from an External database into the current datamodel. However, with the potential of previous residents leaving and new people joining, the assigning of data would require two local counters, so that (for example) the data of Person 5 from last interview could be credited to the same person even though she was now Person 2 in the current wave.

When reading data from the External file, the Blaise would focus only on those individuals designated to household number of the case it was processing. Those passed over in this case would be processed when a case had been spawned for the relevant household number.

Figure 4: Method and Blaise Code for only feeding forward for relevant persons within household

Method:

In RULES section,

- create a FOR..DO loop, arrayed to the maximum household size
- the Local controlling the FOR..DO loop, designates the record read from the External database
- create a second Local which will designate the record number in the current datamodel to which the data will be written; initially set this Local to zero (0)
- for the first element of the array, if the person is designated to the household number of the case, set second Local to 1, otherwise keep it at zero (0)
- for each subsequent element, if the person is designated to the household number of the case, the second Local is incremented by 1, otherwise it retains the value it previously had

Example Code:

```
FOR i:=1 TO 16 DO
  IF (i= 1) THEN
    j:=0
    IF (QTFilter.QBFilter[i].MemHH=QID.HHHold) THEN
      j:=1
    ELSE
      j:=0
    ENDIF
  ENDIF
  IF (i>1) THEN
    IF (QTFilter.QBFilter[i].MemHH=QID.HHHold) THEN
      j:=j+1
    ELSE
      j:=j
    ENDIF
  ENDIF
```

For example, in Figure 5 (below), at Wave 1 the sampled address had five resident eligible people who were interviewed, these are numbered 1-5 top to bottom, left to right. At the following interview (Wave 2), person 1 had died, persons 3 and 4 continued to remain resident at the original address, whereas persons 2 and 5 had moved out together to a new address. At Wave 2, the original household would now be divided for processing into the following groups or ‘households’.

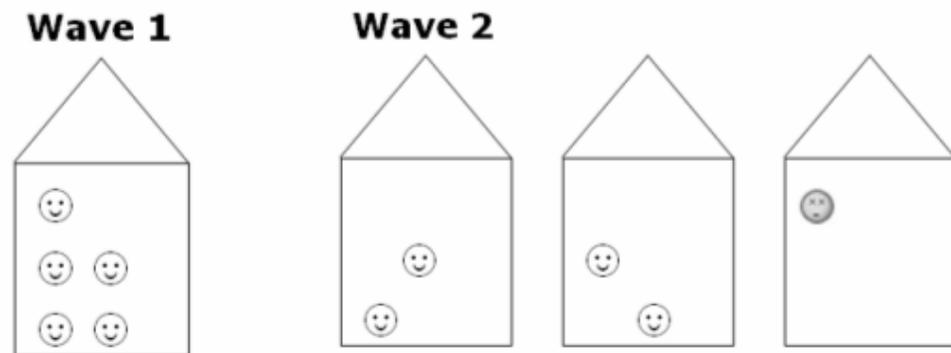
Household 1: Person 1 (Previously Person 3 last Wave) – Resident
 Person 2 (Previously Person 4 last Wave) – Resident

Household 2: Person 1 (Previously Person 2 last Wave) – Mover
 Person 2 (Previously Person 5 last Wave) – Mover

Household 3: Person 1 (Previously Person 1 last Wave) – Ineligible

The process just described will have processed two people from last Wave (household members 3 and 4) for Household 1. The issue that remained was how to ensure interviewers spawned new households in order to process the remaining people.

Figure 5: Pictogram for use with example (above) of how movers and ineligible persons are designated to separate households



3.5 The paradox of referencing the database as an External within its own datamodel

The concern was that, having interviewed those respondents still resident at the original address and coded off the case, there were no checks to ensure that an interviewer spawned new households in order to process the remaining respondents who had moved or become ineligible. Even if they remembered to spawn new households, a method needed to (1) apply the filtering from the first household and then (2) feed forward data from the previous interview appropriate to the people of these spawned households.

Spawned households would be stored as separate records within the same database as the original case. For one of these to be able to read the filter information of Household 1, the database of those cases needed to be used as an External file within the datamodel. This is possible within Blaise, and the metadata and data would simply need to be specified within the datamodel’s Uses and Externals sections, as with any other External file.

The only problem this posed was that the datamodel could not be compiled until the *.BMI metadata file existed, which of course did not exist until the datamodel was compiled. The answer was to compile the datamodel twice over. For the first compilation, the references in the Uses and Externals sections to the datamodel's own database were commented out. Having compiled the datamodel minus these references, a *.BMI metadata file then existed which could be placed in the appropriate directory in order to recompile with the references previously commented out now commented back in.

With the option of the spawned household now able to read the record for Household 1, it was possible to assign the whole of the data from the filter block Household 1 to the spawned household, allowing to same method as above to be used to process the persons designated to that 'household'.

Figure 6: Method and Blaise Code for reading and feeding forward data from Household 1's Filter block

```

Method:
In RULES section,
    • If Household number is greater than 1 (i.e. it is a spawned household),
      then read the case's database for the data line for Household 1
    • Read the line, and keep and assign data from Household 1 to current
      household

Example Code:
ELSEIF QID.Hhold>=2 THEN
  IF CaseData.SEARCH(QID.Area,QID.Address,1) THEN
    IF CaseData.RESULT=0 THEN
      CaseData.READ
      CurStat.KEEP
      MovedWth.KEEP
      ROHM.KEEP
      InOut.KEEP
      DateMov.KEEP
      Countres.KEEP
      Actions.KEEP
      MemHH.KEEP
      CurStat:=CaseData.QTFilter.QBFilter[i].CurStat
      MovedWth:=CaseData.QTFilter.QBFilter[i].MovedWth
      DateMov:=CaseData.QTFilter.QBFilter[i].DateMov
      Countres:=CaseData.QTFilter.QBFilter[i].Countres
      DateDth:=CaseData.QTFilter.QBFilter[i].DateDth
      PID:=CaseData.QTFilter.QBFilter[i].PID
      KnowDet:=CaseData.QTFilter.QBFilter[i].KnowDet
      Add1:=CaseData.QTFilter.QBFilter[i].Add1
      Add2:=CaseData.QTFilter.QBFilter[i].Add2
      Add3:=CaseData.QTFilter.QBFilter[i].Add3
      Add4:=CaseData.QTFilter.QBFilter[i].Add4
      Distrct:=CaseData.QTFilter.QBFilter[i].Distrct
      PstTwn:=CaseData.QTFilter.QBFilter[i].PstTwn
      PostCd:=CaseData.QTFilter.QBFilter[i].PostCd
      SurePstC:=CaseData.QTFilter.QBFilter[i].SurePstC
      TeleNumb:=CaseData.QTFilter.QBFilter[i].TeleNumb
      EmailAdd:=CaseData.QTFilter.QBFilter[i].EmailAdd
      Actions:=CaseData.QTFilter.QBFilter[i].Actions
      MemHH:=CaseData.QTFilter.QBFilter[i].MemHH
    ENDIF
  ENDIF
ENDIF

```

Additionally, this ability to read its own database allowed for greater control in the back-end Admin section of the questionnaire. Counts of household member by category – resident, mover (divided into each subcategory type), and ineligible – were computed in the filter block.

Not only did these allow for the status and outcomes of individuals and cases to be automatically computed in the Admin section of each case (original and spawned households), it allowed to check the status and outcome of related cases. This provided the option to disallow a household from being transmitted back to the office until households expected to be spawned (on the basis of the data in the Filter block) had been opened and completed.

It also allowed for messages to be displayed, against interviewers' cases within their Casebook Pending Tray, highlighting that additional households needed to be spawned and the total number expected.

4. Conclusion

Within ONS, the Blaise Mantra has always been, "Keep it simple". The Filter/Rotation method introduced on the GHS and now becoming standard for other longitudinal surveys at ONS is far from this sentiment.

Though each part of the method uses basic Blaise functionality, as a whole the complex code created problems for researcher-programmers not only in getting to grips with the processes being undertaken but (more so) in the impacts these had later in the questionnaire (particularly within the back-end Admin sections).

In the rotation/filter method of GHS, and in some of the developments considered and introduced for the new Integrated Household Survey (IHS), ONS is now reconsidering some of the boundaries it had previously placed on how it uses Blaise. In aspects like IHS's consideration of Prepare Directives, this was part of the continual review of the usefulness of aspects of Blaise's functionality not currently utilised.

On GHS, however, some of the features, such as checks between cases, would normally have been undertaken outside of Blaise. A Blaise solution was found, however, in order to provide a survey requirement in the light of time and people resources being unavailable from our usual Information Management providers.

With the complex inter-relationships across various blocks within this method, changes need to be thoroughly tested as, though they may perform the action required in the block in which they occur, the change may ricochet with unexpected results in later blocks using these variables. Minor changes in routing or computation of a variable in the Filter block can, if not worked through blocks dependent on it, lead to incorrect outcomes being computed or false errors and messages being encountered by interviewers.

As with most Blaise writing, it is ultimately down to a balance between gains and losses. The cost for a survey in producing Blaise code for a complicated requirement may be too heavy for an organisation to bear. Manners and Green (1996) went so far as to state, "We contend that clever code ... is a bad idea". Admittedly, this was in the context of "doing something in a non-standard way to shorten the code required", however their conclusion may still apply in this case, that, "If there is no discernible difference to interviewers and respondents in questionnaire performance between clever code which only programmers can understand and simple (even if redundant, lengthy) code which everyone can understand, one might argue that the latter type of code is more efficient."

ONS is currently undergoing organisational change, following the establishment of its Head Office at its Newport site in Gwent (South Wales). With all social survey posts transferred or transferring to this office by March 2008, the Blaise-knowledge pool of its new researcher-programmers is still relatively young and inexperienced. As Gatward (2007) highlights reliance on the expertise of a few (which more complicated code promotes) makes programming more susceptible to organisational change. In meeting its new challenges, ONS may need to reconsider its traditional approach on survey-specialist Blaise authors, working more in the hybrid model as detailed by Gatward.

From our experience, Blaise proves itself as a robust program capable of meeting the challenges that social surveys throw at it; the question that remains is the balance between what is possible and what is desirable.

5. References

Brown, A., Hale, A. and Michaud, S. (1998): Use of Computer Assisted Interviewing in Longitudinal Surveys, Chapter 10 of *Computer Assisted Survey Information Collection* (Edited by M.P. Couper et al), pp.185-200.

Fiacco, A. (2007): The ONS Integrated Household Survey: The story so far. To be presented at the 11th International Blaise Users Conference, 2007, Annapolis, September 2007.

Gatward, R. (2007): Authoring Blaise Questionnaires – A Task for the Survey Specialist or an IT Programmer? To be presented at the 11th International Blaise Users Conference, 2007, Annapolis, September 2007.

Jäckle, A. (2006): Dependent Interviewing: A Framework and Application to Current Research, University of Essex, UK, Presented at Methodology of Longitudinal Surveys International Conference, 2006, Colchester, July 2006.

Manners, T. and Green, H. (1996): Blaise III: Who should do what? Internal Social Survey Division (ONS) paper, July 1996.

Exploration of Blaise Instrument Generation from Metadata

Fred Wensing, Juanita Pettit, Australian Bureau of Statistics

1. Introduction

Blaise instruments consist of source code which contain all the descriptive and definitional information about each data field (or question), as well as relationship information between data fields, such as sequencing, edits and derivations. Combined with layout attributes, the compiled source code becomes an instrument which can be used interactively or in batch mode to collect data, edit data, display data and apply amendments.

Instruments are often scripted from specifications, which define the instrument content, data flow, derivations and constraints (edits) on collected data. The specifications from which instruments are prepared can be considered to be the detailed metadata which defines the instrument.

The task of scripting an instrument involves transcribing the metadata content of each data element, adding various presentation standards and conventions, as well as program logic to define the instrument flow and behaviour.

If sufficient rigour were applied to the specification metadata, would it be possible to build a system that could generate the source code for Blaise instruments? This question is explored in this paper. It commences with a detailed analysis of the elements which make up an electronic questionnaire, and leads into the development of a prototype facility which is expected to deliver Blaise instruments from metadata.

2. Context

Blaise has been used at the Australian Bureau of Statistics (ABS) for the development of electronic questionnaires for population surveys since 1995. Since that time, virtually all population survey questionnaires at the ABS have been converted from pen-and-paper mode to computer assisted interviewing (CAI).

The early development of processes to support CAI for population surveys at the ABS had a focus on Case Management (Wensing and Martin, 2001) and defining standards for screen layout (Wensing, Barresi and Finlay, 2003). These were obviously essential for the successful implementation of CAI and took priority over any plans for generation of instrument source code.

For instrument preparation, early emphasis was placed on conventions and programming standards to ensure that there was consistency between instruments, and to ensure that the source code was well managed. The Blaise source code was mostly scripted by hand but drew on standard instrument shells and shared modules. Some developments were undertaken to create utilities to assist with instrument assembly and release (Wensing, 2004) but otherwise there was no generation of source code.

To support the transition from the pen-and-paper forms used in population surveys to electronic forms, a repository was developed to record the specifications of data items and survey questions in a structured format. The main purpose of the repository, known as the Survey Development Tool, was to organise the specifications and make them readily available to all persons working on the survey. The repository was widely used but remained rudimentary, with no particular features for generation of source code. A replacement facility is currently under development and is expected to provide some support for code generation in the future.

Since 2003, Blaise instruments have also been used in business surveys at the ABS for computer-assisted telephone interviewing (CATI) for data collection and sample maintenance (Farrell, 2006). More recently, Blaise has also been used in business surveys to provide a mechanism to apply edits to collected data and to provide an interface to assist with the investigation and resolution of failed edits (Wensing, 2006).

The specifications for the data elements to be included in business survey instruments are prepared in database facilities or spreadsheets. In the same way as for population surveys, the specification facilities for business survey instruments did not include features for generation of source code. Once again, the Blaise source code was mostly scripted by hand, drawing on standard instrument shells and some shared modules.

Over the past decade the ABS has developed elements of a Corporate Metadata Repository which will provide a central store containing the definitions of the data elements used in all statistical collections. The repository is largely based on *International Standard ISO/IEC 11179: Information Technology - Metadata Registries*.

In conjunction with the developments associated with the Corporate Metadata Repository, a project was sponsored to examine the feasibility of supporting Blaise instrument code generation from data element definitions and other metadata. An expected output of the project was a prototype system and explanatory material to prove the concept of generating Blaise instruments from metadata. Aspects of that project and prototype system are described in this paper.

3. Why generate instrument source code?

The act of scripting a Blaise instrument involves taking the detailed specifications, assigning names and types to field definitions, transcribing the specified texts into the field definitions, adding text formatting, and then adding structure, sequence logic, derivations and edits. The work involves knowledge of the Blaise language syntax, knowledge of local programming practices and code storage, familiarity with instrument design and the systems into which it will be deployed, as well as some understanding of the subject matter being covered.

The main reasons for contemplating the generation of source code are:

- to take advantage of infrastructure developed to manage specifications;
- to avoid duplication of effort;
- to minimise the possibility of error;
- to give more control over the content to survey designers;

- to ensure that specification and instrument are consistent with each other;
- to maximise reuse of data element specifications;
- to enable speedy update of an instrument after changes to specifications;
- to facilitate the implementation of standards; and
- to save time.

Code generation will also free the survey expert to concentrate on content and design issues rather than the technical scripting process.

The concept of Blaise instrument code generation is not new and other agencies are also working on this. It could be that the generator just does the initial copying of static text into an instrument structure with further change(s) being made directly to the source code (Egan, 2003). Alternatively, the generation could be achieved by using a specification interface which guides the instrument development one question at a time (Vreugde, de Groot, and van 't Hof, 2003). A more systematic approach would be to have a metadata store which records the attributes of many data elements and provides an interface to enable users to select required elements and generate the code for any instrument (De Bolster, 2004). It is this latter approach which more closely aligns with metadata developments at the ABS.

4. Analysis of questionnaire metadata

In order to understand the complexity of instrument code generation, analysis was carried out to define the full extent of metadata that could be needed to support a questionnaire. This analysis was based on current data modelling for a Corporate Metadata Repository supplemented by current practices in questionnaire construction for both population surveys and business forms and Blaise instrument development practice.

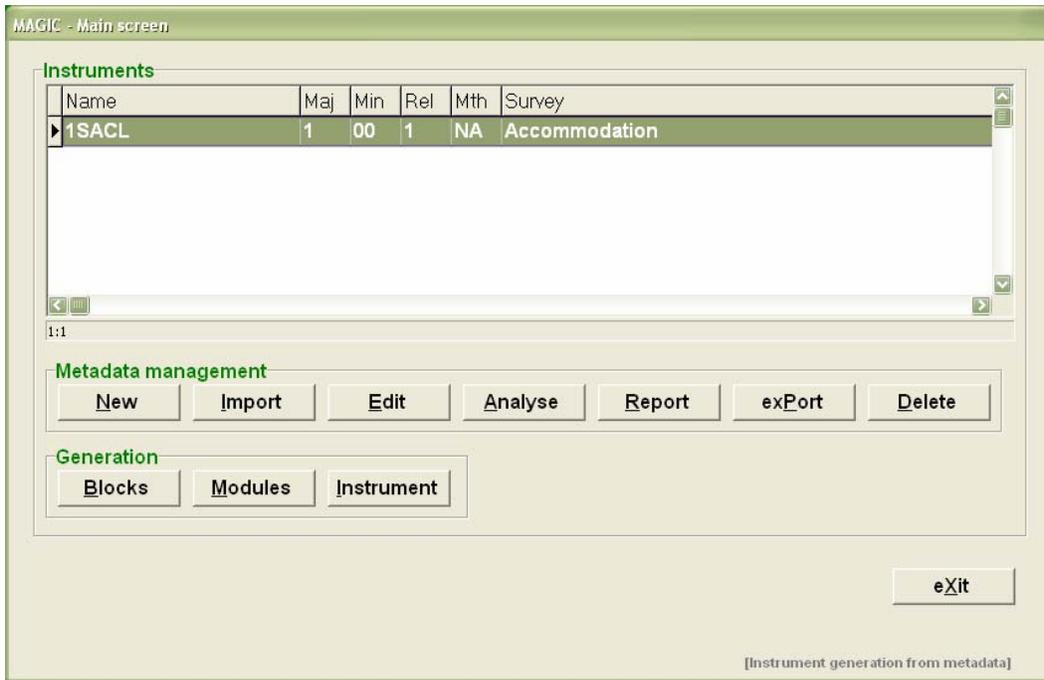
Given that one objective of the project was the generation of a Blaise instrument, a logical starting point for gathering the metadata was the Blaise datamodel and all its elements. This approach also ensured that the Blaise elements and behaviour were accommodated.

Figure 1 shows the data model developed to represent the basic metadata needed to display the questions for an electronic instrument. The basic metadata is restricted to the data elements, questions, text construction, type definitions, response sets and display features. Control events, such as conditional routing, derivation and the application of edits are to be given more attention in the next iteration of the analysis.

All questions and other fields in an instrument are related to Data Elements which are expected to be registered in the Corporate Metadata Repository. Due to ABS questionnaire design practice, particularly for population surveys, the main building block of a questionnaire (or instrument) is the question module. The use of question modules makes it easier to define self-contained sets of fields with their own internal relationships. A question module often translates to a block of source code in the Blaise instrument making this a useful technical distinction as well.

metadata would at least assist in the instrument preparation process through this system. Figure 3 shows the main control screen for the MAGIC prototype.

Figure 3. Main screen of the prototype code generator



The main underlying feature of the prototype system was a table of metadata element definitions. Each row of this table corresponds to one instrument element which in turn may relate to a real data element or to a text object that is displayed on the instrument screen. The columns of the metadata table identify the various attributes that describe each element. Figure 4 shows a list of some of these attributes.

Figure 4. List of attributes recorded for each instrument element

Attribute	Description
System name	A unique system name for the data element
Description	Description of this element (optional)
Question number	Question number (text)
Element type	Type of data element (free text, containing key words relating to type)
Element code	Code letter(s) representing the type of element (e.g. Positive integer; Enumerated; Date etc)
Type specification	Block specification for this type
Minimum value	Lowest value for numeric data
Maximum value	Highest value for numeric data
Category definition	Category definition to use with this data element
Display method	The display method to be used for this element: Ask, Show, Keep
Column	The main column to be used for this element
Lines	Number of line spaces to provide for text
Block name	The block presentation to be used for this element (derived unless a selection is made)
Prefix	Unit of measure to be used as a prefix

Attribute	Description
Suffix	Unit of measure to be used as a suffix
Header	Header for question text
Main text	Main text for the question
Instruction text	Instructions to be displayed
Help text	Help text to be displayed
Alternate text	Alternate text to be used for question

In order to manage the instrument metadata table some tools and functionality were added to the prototype MAGIC facility. These included:

- interface to record the identity and main characteristics of a new instrument;
- utility to import metadata from a comma-separated text file, with flexibility to identify which columns relate to particular attributes;
- ability to edit the attributes for individual questionnaire elements and make corrections or adjustments (see Figure 5);
- utility to analyse the questionnaire elements and produce a summary report of the key attributes;
- utility to export metadata to a comma-separated text file.

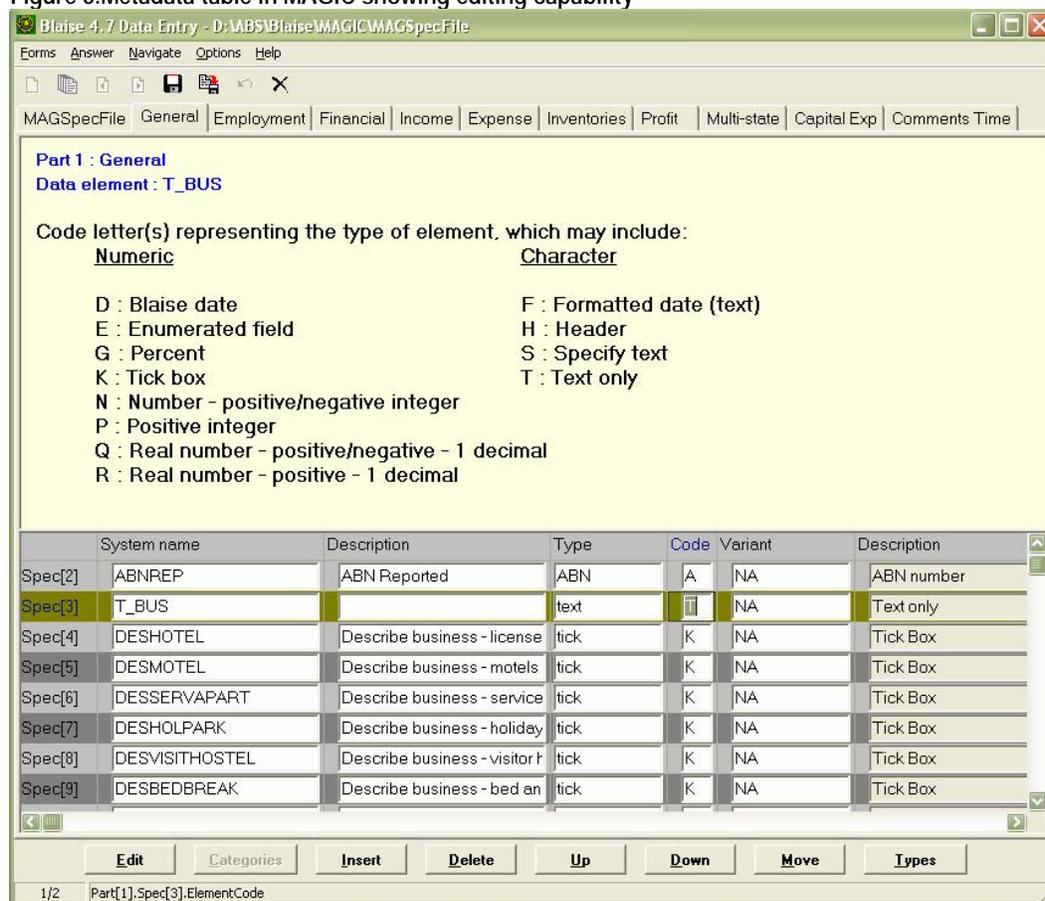
Initial experimentation with the prototype facility looked at supporting the production of business survey instruments to be used in data editing. There were a number of reasons for this:

- the instrument structure used for business survey instruments was well defined, consisting of standard control fields, and a number of sections consisting of a table structure and data elements in columns and rows;
- a shell instrument containing standard control fields and basic operational source code already existed, providing a good starting point for instrument generation;
- question texts in business survey instruments tend to be static (as for paper forms);
- the order of questions in many business survey forms do not require conditional routing to be applied, making it possible to postpone dealing with this issue;
- the specifications of business survey data elements were often prepared in a spreadsheet making it easier to import the metadata.

Requirements for CATI instruments were investigated but project resources were exhausted before generation of CATI instruments could be included in the prototype. A shell instrument was also not available for immediate use as was the case with editing instruments.

A complimentary tool to build template blocks for use in business survey instruments was linked to the MAGIC prototype so that the operator could define new (template) blocks if required. This tool, also written in Maniplus, was developed previously and delivers blocks of code that can be used to display a pattern of data elements in a 'row' on the screen. When applied, each block uses parameters to accept text strings that are part of the row display. Such blocks can therefore be used multiple times in the same instrument.

Figure 5. Metadata table in MAGIC showing editing capability



Once the metadata has been examined and adjusted, and required template blocks are defined, the three-stage generation process can be activated. Firstly, the required template blocks are generated, then the source code for the sections, and then the source code for the main instrument. The last stage draws on the above-mentioned shell instrument containing the standard control fields and basic operational code and inserts the additional source code to identify and activate the sections of the instrument. This final step also makes use of techniques that were previously developed to assist with instrument assembly at the ABS (Wensing, 2004).

6. Discussion and conclusions

The MAGIC prototype has been tested by current instrument programmers at the ABS and has been shown to successfully generate the Blaise source code for business survey (editing) instruments from metadata. It was possible to do this by importing the original specifications (prepared in a spreadsheet) as metadata into the MAGIC prototype. Through the iteration of analysis, adjustment of the metadata and activating the generation steps, successive versions of the same instrument could then be produced. An important feature of the prototype facility was the ability to modify and manipulate the metadata.

The immediate benefit of using code generation, even with this prototype, was the productivity gain for preparation of a first-cut instrument. Furthermore, the ability to

regenerate the instrument following changes to the metadata was also seen as a time-saver.

While the generated instruments were operational, they represented only about eighty percent of the finished product. Additional hand-written code needed to be added in order to provide the edit functionality which these instruments required. It is intended to add functionality to the MAGIC prototype to enable the hand-written code to be extracted from an existing instrument and transferred to the next iteration of that instrument. This functionality will once again draw on the techniques previously developed to assist with instrument assembly (Wensing, 2004).

The formatting of text for screen presentation was largely handled through the application of ABS standards and conventions for screen layout incorporated in the Blaise mode library component. The application of standard formatting was assisted by ensuring that the question text was broken down into small enough parts (or subtexts) each of which have their own standard format. The only way to add text highlighting to particular words within a question, however, was to use the Blaise language mark-up symbols within those texts which may be a problem if the metadata is to be used for other purposes such as the production of paper forms. This problem may be alleviated when the Blaise software enables the use of more conventional mark-up tags (e.g. HTML) within instrument texts.

Although the prototype only supports one mode of instrument at this time, namely editing, consideration has been given to the extent that instruments used in other modes of operation (e.g. CATI) may be generated from the same metadata. While the general layout aspects of different modes could be handled by changing the instrument structure and some of the formatting components, the mode differences may in fact require different wording or question format for the same data element (Pierzchala, 2006). Conversion of questions from an editing instrument (based on a self-administered paper form) into questions that are suitable for CATI is not straight-forward (Farrell, 2006) and will therefore work against the hope of using the same metadata.

There are some other aspects of instruments such as conditional routing and context-specific text fills that have not been tackled in the development of this prototype. These aspects are important for CAI and CATI instruments and will also need to be explored fully before such instruments could be generated.

In conclusion, the development of this prototype facility has demonstrated that it is possible to generate basic instrument source code from appropriately defined metadata, even though some hand-written source code may need to be added. The generation of instrument code is assisted by having standardised instrument designs and some pre-programmed questionnaire elements. A good, flexible metadata management facility is also fundamental to the success of any code generation system. Handling the construction of instruments for different modes of operation and conditional routing will be the next challenge.

7. Acknowledgements

The authors would like to acknowledge the ABS Forms Consultancy Group who sponsored and provided valuable input to the prototype development.

8. References

- Wensing F., Martin B. (2001): Case management for Blaise Using Lotus Notes, Proceedings of the 7th International Blaise Users Conference, Washington DC, USA, September 2001.
- Wensing F., Barresi J., Finlay D. (2003): Developing an optimal screen layout for CAI, Proceedings of the 8th International Blaise Users Conference, Copenhagen, Denmark, May 2003.
- Vreugde C., de Groot J., van 't Hofst C. (2003) : Blaise Survey Generator, Proceedings of the 8th International Blaise Users Conference, Copenhagen, Denmark, May 2003.
- Egan M. (2003): CodeBuilder2 – An Initial Coding Tool, Proceedings of the 8th International Blaise Users Conference, Copenhagen, Denmark, May 2003.
- De Bolster G. (2004): Generating Blaise with Blaise, Proceedings of the 9th International Blaise Users Conference, Gatineau, Québec, Canada, September 2004.
- Wensing F. (2004): Instrument assembly, documentation and release, Proceedings of the 9th International Blaise Users Conference, Gatineau, Québec, Canada, September 2004.
- Farrell E. (2006): Design and testing of CATI instruments for Business Surveys, Proceedings of 2nd Telephone Survey Methodology Conference, Miami, Florida, USA, January 2006.
- Wensing F. (2006): Using Blaise to Apply Edits to Data held in an Input Data Warehouse, Proceedings of the 10th International Blaise Users Conference, Arnhem, Netherlands, May 2006.
- Pierzchala M. (2006): Disparate Modes and Their Effect on Instrument Design, Proceedings of the 10th International Blaise Users Conference, Arnhem, Netherlands, May 2006.
- <http://www.metadata-stds.org/11179/> : ISO/IEC 11179, Information Technology - Metadata Registries.

Using Blaise 4.7 for data entry, checking, error reporting and transforming for further processing

Krstanova Orhideja, SSO R..Macedonija, R.Macedonija

I like to tell you about our experience in using BLAISE survey processing system. We use BLAISE for data entry and control about a year. During that period all new data entry programs were made in BLAISE. Some of the data entry applications for the ongoing surveys were rebuild in BLAISE too. Finding it very quick and convenient especially for large databases our intention is to use it for major part of the surveys we provide.

The things are set like this: The application for certain survey in BLAISE is settled on server in special folder for that survey. Operations performed are: data entry with online control, batch control and reporting errors using manipula, corrections and transforming data using cameleon, in formats suitable for further processing. Regarding that our database is DB2 for all surveys, and that so far we don't use direct communication between DB2 and BLAISE, the data are always transformed in text format so that it would be appropriate for further manipulation or maintain. Sometimes data is transformed in SAS format according to the needs. All these operations are performed thru interface built in Visual Basic 6 and are partly automatic. But we have vision and work on it to make environment, which will provide possibilities for whole processing data to be more automated.

1. Introduction

The first introduction with BLAISE in SSO of Macedonia was about nine years ago. The very first BLASE data entry application was built for Labor Force Survey and with the professional assistance given by the Danish-Finish Consortium.

For many years, this was the unique data entry application in Blaise software. Last few years IT-sector in SSO of Macedonia is looking for a way for more efficient and up to date work in fields like data entry, editing, maintaining, metadata, documentation and so forth. In that context, BLAISE was found appropriate because it allows quick and consistent work especially with large databases. Nearly all the new data entry applications in the last year were built in BLASE.

2. Data entry application

Things are organized like this: there is a folder on the server named with the same name as the survey, the application was built for. That folder contains VB-6 application that serves like interface through which the person that will perform the data entry can reach the application.

The applications are made in blaise data files were code contains standard blocks and tables that corresponds with the questionnaire design.

In most data entry applications data controlling is provided on line. Depends on the construction of the questionnaire for the certain survey the control is performed within the table rules or in datamodel rules or in both. For most of the surveys there is a data base with some basic information like address, business unique number, classification, and so on. Every observed unit has a unique identification number for easier identification. Through this unique number all data for the unit can be linked.

During data entry to save time and disable errors in the basic information this data are taken from mentioned data bases. Next part of the code is presenting this operation:

```
.
RULES
  ISTRAZ :=('SUM22')
  GOD :=('2007')
  RBR {unique number}
  SOP
  OPST
  NKD

  IF Adresar_SUM22.search (RBR) then
    Adresar_SUM22.READ
    SOP:=Adresar_SUM22.SOP
    OPST:=Adresar_SUM22.OPST
    NKD:=Adresar_SUM22.NKD
  ELSE
    ERROR INVOLVING(RBR) "It is not in the address book"
  ENDIF
.
```

The on line control performs logical and mathematical controls, usually using If...then structure. Part of code as an example is given below.

```
.
P41313

If (P43=2 or P43=3) and (P4131=0 and P4132=0 and P4133=0 and P4134=0 and P4135=0
and P4136=0
    and P4137=0 and P4138=0 and P4139=0 and P41310=0 and P41311=0 and P41312=0
and P41313=0) then
    error involving (P4121)"G25: Ako P43=2 ili P43=3 togas mora da ima odgovor vo koja
zemja i kolkav e ostvareniot odmor "

ENDIF
.
```

```
Or
.
```

```
T1[5]
T1[6]
T1[7]
T1[8]

If (T1[5].k1 <> T1[6].k1 + T1[7].k1 + T1[8].k1) or (T1[5].k2 <> T1[6].k2 + T1[7].k2 +
T1[8].k2) or (T1[5].k3 <> T1[6].k3 + T1[7].k3 + T1[8].k3) then
    error "GR 2: vkupnoto e razlicno od zbirot @r ^ T1[5].k1 , @r @r ^ T1[5].k2 , @r @r ^
T1[5].k3 vo polinjata "
ENDIF
.
```

Some times the data have to be controlled with the data from the previous period. If it is possible for these cases, on line control is performed too. As it is shown below during executing data entry for SUM22 survey, certain data are controlled with the SUM22_1 which are data from the previous period.

```

.
Tabela1
Tabela2
Tabela3

If Sum22_1.search (RBR) then
  Sum22_1.READ
  FOR i:=1 TO 30 do
    If Tabela1.T1[i].k2 <> Tabela1.T1[i].k1 - Tabela1.T1[i].k3 +
Sum22_1.Tabela1.T1[i].k2 then
      error "GRESKA : T1[i].k2 e razlicno, vo sporedba so prethodniot mesec "
    ENDIF
  ENDDO
ENDIF

```

Persons that perform data entry are well skilled and familiar with the applications and surveys for qualitative data entry.

3. Checking and error reporting

Although the main parts of errors are prevented by on line control during data entry, sometimes there is need the entered data to be checked more times in different ways. These operations are provided through manipula file type. If the errors that can not be solved by the people that perform the data entry are expected, code for data entry (in BLAISE data model) is a little bit different. If this type of error is expected, previously presented code will look like this:

```

.
T1[5]
T1[6]
T1[7]
T1[8]

If (T1[5].k1 <> T1[6].k1 + T1[7].k1 + T1[8].k1) or (T1[5].k2 <> T1[6].k2 + T1[7].k2 +
T1[8].k2) or (T1[5].k3 <> T1[6].k3 + T1[7].k3 + T1[8].k3) then
  error "GR 2: vkupnoto e razlicno od zbirot @r ^ T1[5].k1 , @r @r ^ T1[5].k2 , @r @r ^
T1[5].k3 vo polinjata "
  greska[2]:=2
endif

```

The difference is in greska[2]:=2 (error[2]:=2) where the errors are defined, then in rules

```

.
RULES
for i:=1 to 7 do
  greska[i]:=0
enddo

```

All this is used in manipula data file for making a report on errors. The report contains the unit number (expressed through unique identification number) in which the error is found and the number of the error. With the unit number and the number of the error, the questionnaire can be easily found and correct the data where it is necessary.

If further processing is performed in ACCESS or have to be input in DB2 database, blaise data is transformed in text file through manipula, sometimes made with cameleon and sometimes separate manipula file is made.

5. Future activities

For now the data entry is performed through an interface built in Visual Basic 6 and it is partly automatically. Data editing, processing and transforming is performed more or less manually. An environment which will provide possibilities for whole processing data to be more automated will be established when some decisions regarding data entry, editing, maintaining, metadata, documentation and so on are made.

At this stage we experience only basic BLAISE software, but we expect and work on to establish surrounding that will allowed us to use the advanced possibilities in favor to more efficient and up to date work.

Contact: orhideja.krstanova@stat.gov.mk

Survey Specifications Management at Statistics Canada

Pamela Ford, Statistics Canada

1. Introduction

Computer Assisted Interviewing (CAI) plays an important role in Survey Collection at Statistics Canada. Over the years, various software packages have been used for CAI instrument development with Blaise being adopted as the standard tool in 1998.

CAI has resulted in increasingly complex questionnaires and survey instrument developers have noted that the preparation of the questionnaire specifications is one of the most time consuming aspects of instrument preparation. Moreover, multi-mode collection, which would require specifying and developing additional instruments, is being viewed as a solution to some of the current collection problems and issues. Under existing practices, adding a second or third mode of collection will result in a significant increase in the total development costs. Consequently, it was determined that the processes and methods used for instrument preparation needed to be significantly revised.

As part of Statistics Canada's Collection Modernization Initiative, the Survey Specifications Manager (SSM) is being developed to facilitate a streamlined specification development process that encourages the standardizing and recycling of questions across multiple modes of collection.

2. Problem Statement

The following items were identified as problems to be addressed within the scope of the SSM project:

- Survey development is currently stove piped often resulting in duplication of specification and development effort.
- Questionnaire requirements are difficult to specify and the specifications process is perceived to be taking too long.
- The current specification process allows for too much change through the development cycle which increases the time it takes to program, test and re-test.
- Survey input file structures are not always clearly specified and validation of such files is limited and often non-existent.
- Survey output from different collection media is formatted differently which increases complexity of subject matter processing systems.
- Not all survey metadata is readily available for dissemination products.

3. Survey Specification Process

It is necessary to formalize a process for the development of survey specifications making use of the SSM application where appropriate. At the time of the writing of this paper, the aforementioned process is far from refined but several key steps have been identified.

3.1 Mode-independent Survey Specification

One of the early steps in the process is the identification of concepts to be measured and the variables and level of detail required for analysis and dissemination. The list of required variables is essentially a mode-independent survey specification. Each variable in the specification is associated with a particular concept which in turn can be attributed to an overlaying theme.

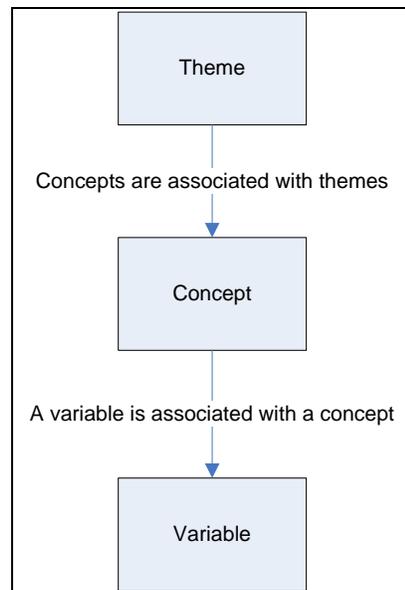


Figure 1 Relationship between Variables, Concepts and Themes

3.2 Survey Collection Requirements

Once the basic content and methodological requirements of a survey are determined it is necessary to assess the available collection options. The following questions might be asked and answered:

- Which collection methods are most suitable given the survey requirements?
- When can collection feasibly take place?
- Can the information be collected by proxy?
- What type of sample frame(s) will be used?
- Will information be fed back from previous iterations of the same survey?

3.3 Survey Instrument Specifications

Once a mode-independent specification and basic collection requirements (Sample frame, mode of collection, proxy/non-proxy etc.) are ascertained, it is possible to proceed with the specification and development of survey instruments.

Survey instruments consist of components each of which address a particular segment within a questioning process. Examples of Social Surveys components include:

- Contact – making contact with an appropriate respondent, determining if the sample unit is in scope
- Household – collecting household-level variables, rostering the household, selecting one or more respondents for which subject matter content will be collected
- Subject Matter Content
- Exit – questionnaire closing (eg. Thank you statements) and coding (registering the outcome of an attempt)

Components, in turn, consist of blocks of related questions with each block collecting data for variables associated with a common concept.

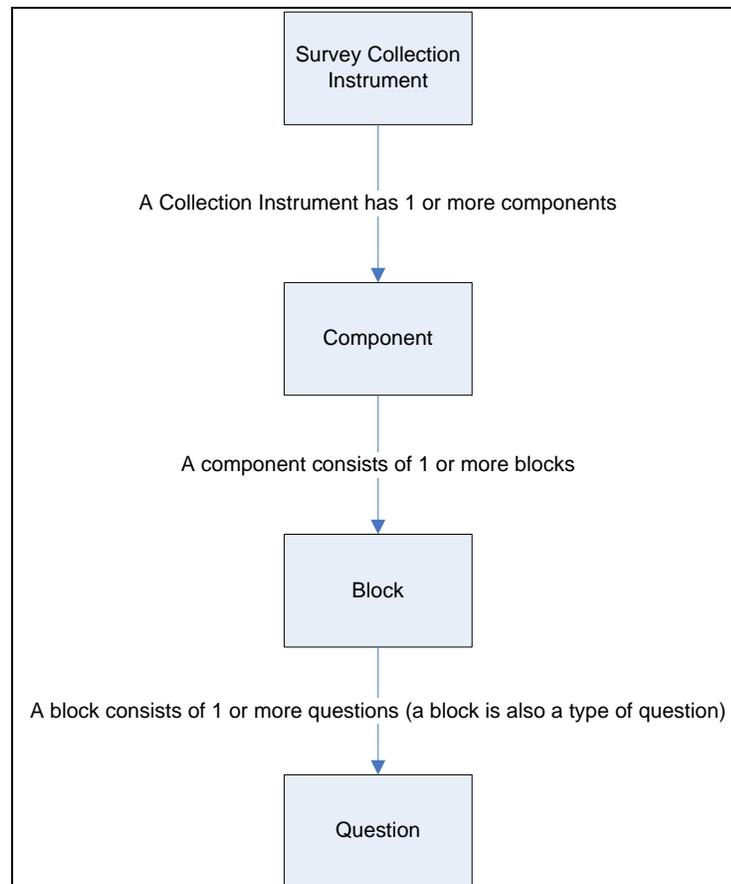


Figure 2 Survey Instrument Design

At this stage, the survey specification process should encourage and possibly even mandate the re-use of existing or standardized question blocks.

3.4 Collection Input

The survey specification process should allow for the identification and definition of collection input other than the survey instruments. This includes but is not limited to:

- Defining the structure of survey input or sample files
- Identifying key (strata) fields to be used in the active management of collection
- Identifying targets for collection at national, local, and stratum levels

3.5 Structured Metadata File

Following collection there is a desire for a structured metadata file including survey instrument metadata to be included with dissemination products. Examples of such metadata might include variable names and descriptions, proxy/non-proxy indicators, date and time of collection for each variable, response/nonresponse indicator by variable, flags to identify if edits had been triggered, etc.

4. Survey Specifications Manager

At the core of the project is the development of the Survey Specifications Manager (SSM) application itself. This involves building a central repository to house survey specifications, instrument source code, and related artifacts with a web-based user interface. There are three key components to the SSM:

- Manage Blocks
- Manage Surveys
- Search

4.1 Manage Blocks

Instrument specifications and code are organized as components or collections of “blocks” with each block consisting of one or more questions with conditional flow logic. One of the goals of the SSM is to facilitate the re-use of blocks across multiple surveys.

The ‘Manage Blocks’ feature is being designed to allow users to create and manipulate individual blocks, outside the context of a survey. This is particularly important for a team of subject matter users who are currently tasked with developing a series of harmonized questions to be used by all social and household surveys.

4.2 Manage Surveys

The 'Manage Surveys' feature will essentially provides users with a means of managing all survey specifications associated with collection. For each survey users will be able to:

- define a mode-independent survey specification
- define survey instrument specifications, making use of existing specifications when available
- define all input associated with the collection of the survey
- produce output in specific formats such as a technical specification for Blaise authors or a DDI-based XML file for survey analysts

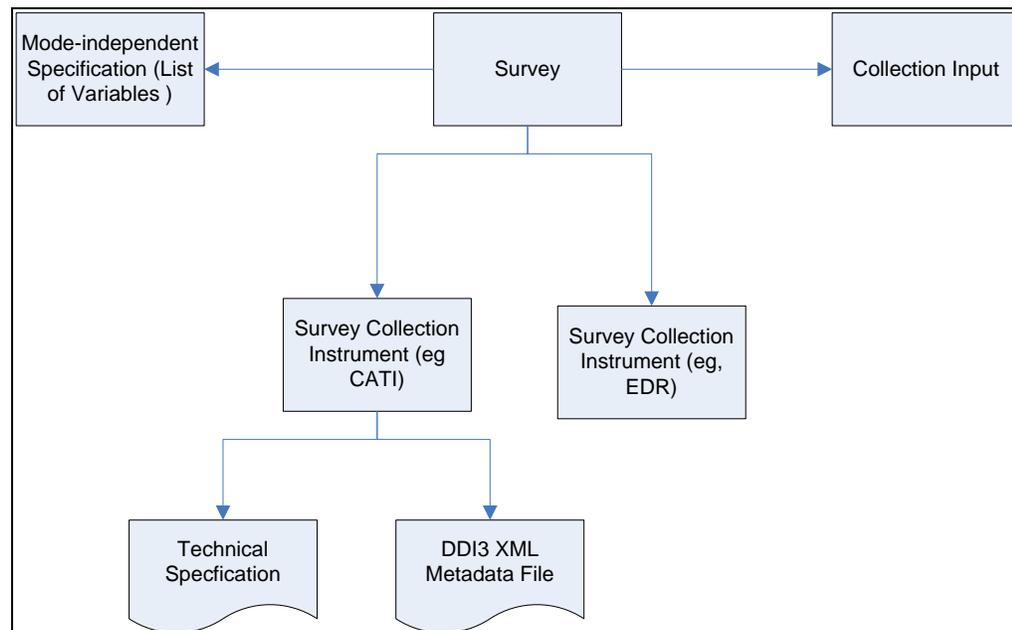


Figure 3 Survey Specification

4.3 Search

The 'Search' feature will allow users to search for questionnaire blocks by various attributes including:

- themes (e.g. Demographics, Health, Education, Employment)
- concepts (e.g. Age, Smoking,)
- variables (e.g. Age of person, Age when person first smoked)
- approval-level (Corporate Standard, Program Standard, Survey-specific)
- survey (e.g. Labour Force Survey, Canadian Community Health Survey)
- question techniques (lookups, mark-all-that-apply etc)

5. Challenges

5.1 Questionnaire Flow Logic

One of the biggest challenges in creating a survey specifications development tool is in facilitating the specification of questionnaire flow logic. CAI has resulted in complex specifications that go far beyond the simple answer-based 'go to' statement. A number of options will be investigated in order to meet this challenge, including allowing for users to specify flow using a graphical interface.

5.2 Historical Metadata

To maximize the benefits of the Survey Specifications Manager, it will be necessary to load historical metadata into the central repository. To facilitate this, plans are being made to create XML versions of historical block and questionnaire specifications which can then be easily imported into the structured SSM database. The Michigan Questionnaire Documentation System (MQDS) is being considered as a means of producing this file until such a time as it is possible to generate XML files from Blaise.

6. Conclusion

The SSM is intending to provide a one-stop shop for the management of all specifications related to survey collection. The expectation is that using this tool within a formal specification process will result in making the overall survey collection process more effective.

7. References

User Guide – Michigan Questionnaire Documentation System (April 2007), Survey Research Center, The University of Michigan.

8. Contact Info

Pamela.Ford@statcan.ca

Analyses of Web Survey Data

Vesa Kuusela, Social Survey Unit, Statistics Finland

1. Introduction

Data capture process of a self-administered questionnaire is inherently different from that of an interview. That has many implications on the settings in which data capture takes place. In a web interview (CAWI), the whole response process differs from that of an interviewer-administered interview. In CAWI respondent answers alone without any assistance or control, and sending the filled forms requires only a click of a button. In addition, the different implementations of web based data collection change the settings of responses process: in a static web questionnaire only a limited amount of checking can be implemented and most of checks are run only when the form is received in the web server. In an interactive questionnaire there is a lot more possibilities for checks and checks are run after a page has been sent and data is stored after each page. In an interactive questionnaire also question order can be defined which is not possible in a static form. The settings of the web based data collection produces such errors in data that do not exist in other modes.

In an interviewer-administered survey the interviewer controls the situation and what is entered in the form. If a respondent agrees to be interviewed he or she usually contemplates the questions and answers. Very rarely the answers given in an interview are completely nonsense. In any case, the interviewer is able to assess the feasibility and reliability of the answers. Breaking a started interview does not happen very often. If a respondent has agreed to be interviewed in most cases he or she does not interrupt answering because of minor discomforts. Consequently, the data obtained from an interviewer-administered data capture – using a well-defined instrument – will be ready for editing and analysis after minor checking. In addition, unit non-response is coded adequately and may be processed using available methods. There may occur item non-response, though. Usually it remains in a moderate level, however.

A web interview differs also from that of a postal survey even though both are self-administered. In a postal survey, like in a web survey, the data capture process cannot be controlled. However, in a postal survey, the forms have to be returned to the agency or company that is conducting the survey and that requires some effort. Sending a static web questionnaire is usually a very simple task and when using an interactive web questionnaire the data is stored all the time requiring no special effort. In a postal survey the returned forms are usually checked before data entry. The validity of the data is assessed (and screened) during checking and also during the data entry from the forms. Therefore in a postal survey, the data to be analysed mostly is composed of answers, which can be regarded as given seriously and with consideration. In a web survey one cannot be as sure of that.

The different response situation has given rise to new types of errors in survey data and that calls for new types of data cleaning methods. The new problem cases should be filtered out or otherwise be taken into account in the data analysis of the survey. Theoretically interrupting the data capture leads to item non-response. It is slightly different from conventional item non-response where values are missing here and there. In an interrupted filling of questionnaire all the answers after the “break point” are

missing. Some respondents are interested in what is asked but are not interested in giving well-considered answers to them. A form that is filled in only to browse through the questions is more problematic. Basically it should be classified as unit non-response but it may require sophisticated methods to single them out from “serious” answers.

The new problems of the data quality have not received much attention, yet. For example Dillman (Dillman, 2007) or Reynolds et al. (Reynolds et al., 2007) don't recognise nor discuss them. It is difficult to know whether there has been any research touching this problem because there doesn't exist many reports. The aim of this study is to find out whether breaking and browsing exists and if yes, to what extent.

2. Material and Methods

The material for this study comes from a survey conducted by an afternoon paper in Finland. On its Internet site was general invitation to readers to fill in a questionnaire on its site. The questionnaire consisted of 20 statements comprising a psychological profile of respondent's emotional intelligence. Statements dealt with the traits of character and respondent was asked to answer whether the trait did not describe him or her at all, did not describe well, described fairly well, described well, or described him or her very well. Many of the statements were contradictory in such a manner selecting the first option (say) for each statement could not be true for one person. For example “I usually plan my activities well ahead” and “I rather improvise than plan carefully”; “I often take risks only basing on my feelings” and “I rely more on facts than feelings”; or “I often show sympathy to other people ” and “I refrain from showing emotions that might impede my performance or my human relations”. However, the formulation of statements was not done to facilitate the screening of inconsistent responses.

The questionnaire was a simple page based form and there was no consistency checking for given answers. The statements were given in blocks (or pages) of four. The “survey” had a general invitation without any identification of respondents. That means that there was no control of who responded or how many times a response was given. Neither the time nor duration of answering was recorded. The only background variables were respondent's gender and age class.

The questionnaire was open for four weeks. A total of 202 411 responses were recorded in that time. Women answered slightly more frequently than men (53.9% vs. 46.1 %). Young respondents, under 25 years, were the largest group (43.4%) and second largest group were young adults between 25 and 44 years (39.2%) and middle aged or older readers responded most infrequently (17.4% of responses). Probably, the demographics are typical to this kind of a CAWI survey but do not represent the population neither the readers of the paper.

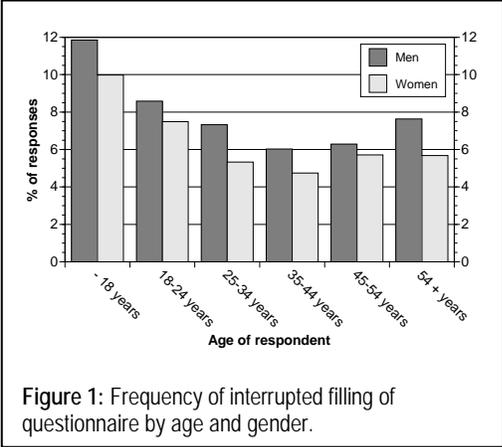
3. Results

The analysis is focussed only on interruption of filling and finding out potential browsing of the questions. This was done using only fairly simple criteria. Very small variation between answers was assumed to indicate that the answering was not based on deliberation. High variation also may show browsing but it is more sensitive to erroneous classification. Also the use of the extreme answer categories only may be an indication of browsing. The variation was analysed using standard deviation, s , as a

criteria. Also “too” small or large mean value, \bar{x} , may indicate browsing as well as the values of the coefficient of variation, $100s/\bar{x}$. More sophisticated criteria could be applied but they were not tested in this study.

3.1. Interruption of interview

An answer to all statements were given in 92.55% of the responses, and in 7.45% of cases the filling was discontinued at some point. Most often the interruption took place after the first block of four statements (4.05% of all responses). As the questionnaire was arranged in blocks of four statements, in most cases the interruption took place after a block or after receiving a new page.

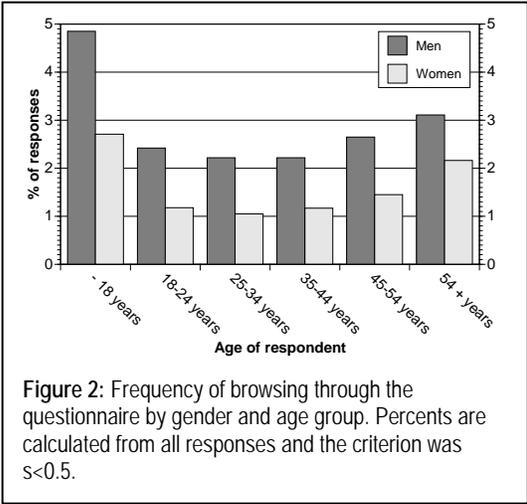


The male respondents interrupted filling of the form more frequently than females (8.25% vs. 6.75%). Also among the young respondents interruptions occurred more frequently than among the older ones (see figure 1). In each age group the interruption frequency of men was higher than that of women. Probably this was partly related to the topic of the questionnaire that can be guessed to interest more women than men.

3.2. Browsing through the questionnaire

Browsing means that the respondent does not answer to the questions after reflecting them or does not indicate his or her true reaction to a statement. Instead, they enter some answer only to move ahead in the questionnaire. Obviously there are many different styles to give answers to reach this goal. For example, selecting always the first alternative or the same alternatives that are vertically (or horizontally) lined up. The other possibility is to select in turns the first and the last option (e.g.). Certainly there are also other styles, but these two styles may be identified using simple statistical criteria: the first one minimises variation and the other style maximises it.

In 1.13% of all responses there was no variation between answers (standard deviation was 0) indicating that all the answers had the same value. When only the complete answers (answer in each of the 20 statements) were analysed, 0.97% of the cases had no variation. This indicates that those who are not really answering to the statements are more inclined also to interrupt the filling. In most of the cases without any variation the selected answer category was the



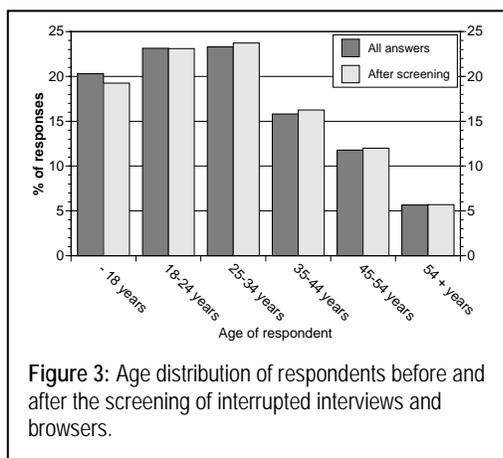
one in the middle. Also cases where the only selected answer category was the first one were common. Logically or psychologically neither of them can be true.

Using slightly wider criterion ($s < 0.5$), obtained when one or two of the answers were different, led to exclusion of 2.17% of all cases and to exclusion of 2.01% of completed cases. This criterion may have led to omission of few “real” answers, but it is not probable. The browsing showed similar age and gender pattern as interruptions. Men were browsing much more frequently than women and young respondents were browsing more often than older ones. In figure 2 is shown the frequency of browsing. Percents were calculated from all respondents using the wider criterion for variation (i.e. $s < 0.5$). However, the pattern was similar if percents were calculated only from completed cases or using the stricter criterion.

Using “too” large standard deviation as a criterion seemed to single out also valid answers. With slightly different formulation of statements had enabled also this criterion. In this case it had required adding some logical criteria to screen only implausible answers. On the other hand, coefficient of variation (CV) screened some illogical cases: by using criterion $CV < 15$ for coefficient, 2.23% of all responses were screened and 1.77% of complete responses. Nearly all of the screened cases were such that all given answers were close to the centre of the scale. Some of these cases would have been screened also by using the standard deviation criterion, but not all.

The mean value of answers could also be used as a criterion: too small or large values indicate the greatest part of answers is close to either end of the scale. However, it proved that mean is influenced by extreme values too much to be reliable.

The different criteria, which were tested here, overlapped to some extent. Using the strict criteria for screening browsers and including only complete cases left 89.67% of the cases. It is probable that very little, maybe none, of truthful answers were screened out. It is more probable that all invalid cases were not screened out. Using slightly wider criteria decreased the number of accepted responses considerably but at the same time the number false negatives seemed to increase relatively fast.



The applied screening criteria left false positives in the data but it is not known how much. In order to minimize the probability of false positives and false negatives more sophisticated screening methods had been needed.

In Figure 3 is shown the influence of the strict screening on the age distribution of respondents. After screening the proportion of young respondents decreased and the proportion of middle-aged respondents increased. Additionally, the proportion of women increased from 53.87% to 54.61%.

4. Conclusions

In a web survey, only partly filled forms, or interrupted interviews, is a new phenomenon that occurs only rarely in interviewer-administered surveys or in postal surveys. An interview, once it has been started, is rarely interrupted, and in a postal survey partly filled mail questionnaires usually are not sent or the data is not entered. Web based data capture takes place in different circumstances, which result in interruptions in the filling in the questionnaire. With a static web questionnaire this is a lesser problem because there is a separate “Send” button, which has to be clicked. In an interactive web questionnaire interruptions may become a problem because answers are stored all the time more or less automatically.

The aim of this study was to find out how difficult problems breaking an interview or browsing questions may pose, but the results of this small study should not be generalized too quickly. The analyzed web survey was an unrestricted self-selected survey, close to an entertainment poll, with a general invitation. In a CAWI survey carried out by research agencies and with a more “serious” topic, respondents both open the survey with more seriousness and also pertain more to thought when they answer the questions. However, both of these phenomena probably are present in all web surveys in smaller scale. Based on this study, it seems that young respondents and men are more inclined to interrupt a web interview.

The main reasons leading to interrupted interviews are loss of interest and frustration. They originate partly from technical factors but more probably they originate from the questionnaire itself: Unpleasant layout, badly formulated questions, unfitting answer categories, difficult structure, etc. most probably frustrate some respondents to a point where they stop.

The response process in a web survey is more sensitive than in an interview. A real interview once started is not interrupted because of minor discomforts but that danger always exists with self-administered questionnaire. Careful planning of instruments is of great importance in avoiding interruptions. In a web survey it is even more important than in a postal survey because in a postal survey the forms do not end up in the data file. A problem with interrupted interviews may be that they may remain unnoticed by the checking procedure because there is a record stored although it is incomplete.

Answers, which have been stored after browsing through the questionnaire, are more problematic than interruptions because they are more difficult to locate. In addition, they may cause bias in the results if they are not found out. The magnitude of bias depends on how much browsing has occurred. Basically, responses obtained after browsing through a questionnaire should be coded as unit non-response.

In designing the web questionnaire it is possible to make “traps” that would catch browsers. If it is possible the answer alternatives should be selected in such a style that browsing through a questionnaire leads to inconsistencies between answers. This way they could be identified in data checking or even during data capture. However, it is not certain how well the traps would work out in real situation. In addition, duration of the interview may provide information on how serious the respondent was.

Breaking the interview and browsing through the questionnaire are not completely separate phenomena. A respondent only browsing the questions will stop when he or she notices that the questions are not interesting enough.

It is quite obvious that the data collected via the web needs a thorough checking, much more than interview data and more than data from a postal survey. In addition, the checking procedure has to be designed on different basis than when checking data collected in other modes.

5. References

Dillman Don, A.: Mail and Internet Surveys. The Tailored Design Method. 2nd edition, 2007, Wiley.

Reynolds Rodney, A., Woods Robert, Baker Jason D.: Handbook of Research on Electronic Surveys and Measurements. 2007, Idea Group Reference.

Web Application Stress Testing with Blaise IS

Jim O'Reilly, Westat

1. Introduction

Survey data collection over the Internet is an increasingly important technique. With the release of Blaise 4.8 in July 2007, the Blaise community has a much improved and more powerful system for web surveys. Blaise 4.8 enables development of web questionnaires and administration of surveys across a wide range of requirements—from small to large sized questionnaires and from small to large sized numbers of survey takers.

Central to web survey implementation is providing the web server infrastructure to support peak numbers of survey users. With version 4.8, Blaise has made major improvements to server-side responsiveness, robustness and scalability. An individual web server can support all functions for small to medium applications and user volume. For large scale situations multiple servers can be integrated to support web, database, and rules services suitable for the application.

Determining in advance whether a given server system is sufficient to support a web survey's requirements is critical for planning and configuration. Web stress testing systems are designed to address this important issue. We discuss here methods and applications using a web application stress test system with Blaise IS.

While we present results of stress tests, our main focus is the methods used. The results are highly dependent on the size and complexity of the data models as well as the server platform. In other words, your mileage may vary.

2. Microsoft Web Application Stress Tool

We used the Microsoft Web Application Stress tool (WAS). This is a freeware product distributed by Microsoft. While a legacy product that has not been updated in years, and which Microsoft offers without any support, WAS is a reasonably capable tool that can be adapted to a number of applications. WAS seems to suit Blaise IS testing situations. To obtain WAS, and for instructions on its use, a Google search of “web application stress testing” returns a variety of useful links.

2.1 Using WAS

Here is an overview of the WAS testing process:

- WAS runs on a user workstation and records a web session. All the http traffic of the session—messages generated by the user and sent to the server and the responses returned by the server—are written to the WAS database. The recorded series is called a script.
- To test using a script, one sets parameters—number of users (called threads), test time, and others. When the test is started in WAS, simultaneous web sessions to the server are generated, sending for each the script's recorded messages in sequence after a preset and/or random delay. The time and result of the return

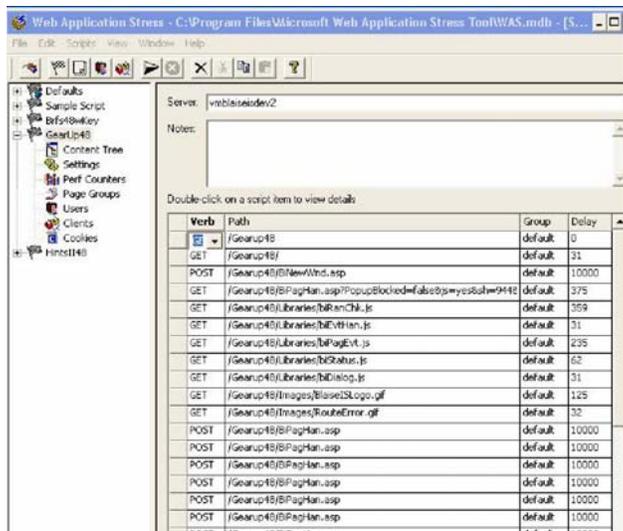
message from server is saved. As a thread is completed for a session, a new one is started until the specified test time is reached.

- Following the test, WAS produces a series of reports on the web interactions-- result codes and counts, time for the requested page to return to the workstation, and more.
- A single workstation can simulate tens or hundreds of web survey users for lengthy periods and provide valuable data on server performance.

3. Blaise IS Stress Test Model

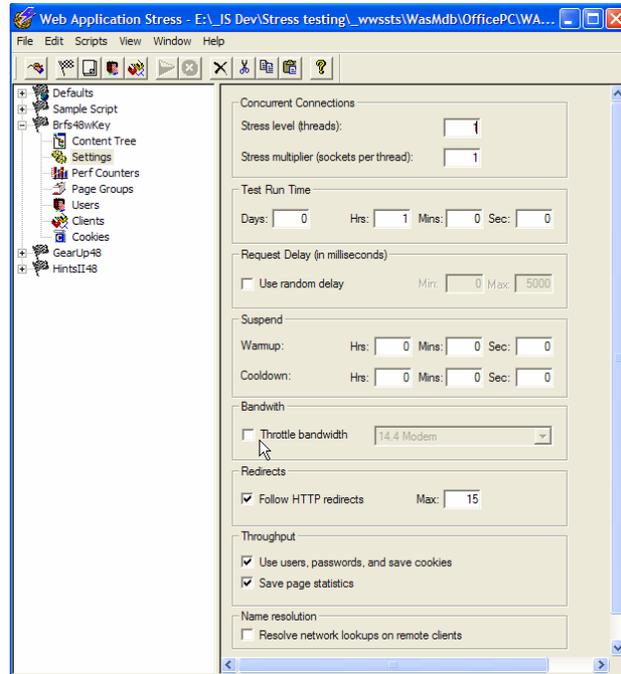
Our testing follows the method used by the Blaise team for web server performance testing reported in the Blaise 4.8 Online Assistant. The key metric is the additional waiting time per page as the number of concurrent users increases. The steps for testing a web application are:

- Record the Blaise IS survey session in WAS.
- In the WAS application, set the script's page delay value to 10 seconds per field on a page for pages that are posts to the biPagHan.asp—the IS page handler.

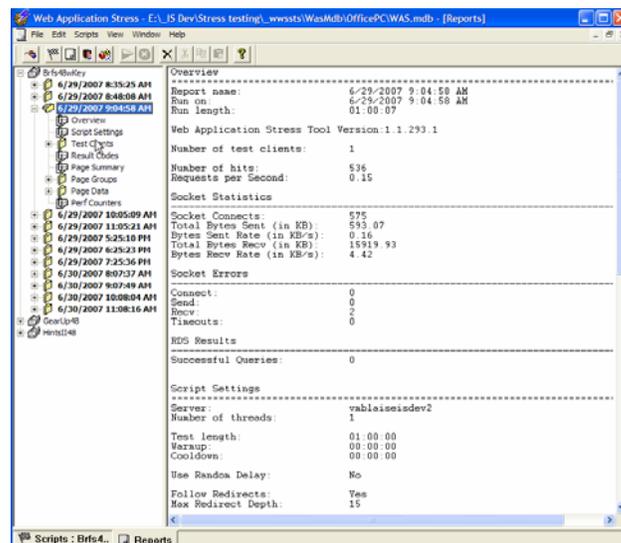


A page with one response field would wait ten seconds before WAS transmits the recorded response to the server. The ten seconds delay per response field is the assumed time required for the average user to read the item, decide on a response, enter the response, and transmit it to the server.

- With the script ready, one sets the parameters for a single-user benchmark test
 - 1 thread (user)
 - 1 hour test time



- After the test is run the report is examined checking that there are few connect errors and recording the socket connects, which is the number of pages received from the server by the browser.



The test time divided by the page count gives the time per page for the baseline single-user situation. For the one hour test of one thread this is 3600 / Pages.

- Re-run the script successively, increasing the number of threads and record the page count and compute the time per page per user $(3600 * \text{Threads}) / \text{Pages}$

3.1 Tests

We conducted a series of tests using three different data models

Table 1: Data models and WAS Scripts

	Fields	Signals/ Checks	BMI size (kb)	IS Pages	Script Pages	Script Pages	Page Delay (s)
Brfs48wKey	41	0	18	20	17	17	22.2
Wes1	640	29	543	83	31	31	23.2
Wes2	370	22	297	221	177	177	11.8

BRFS uses the core items of the well known Behavioral Risk Factors Survey instrument (www.cdc.gov/brfss) of the U.S. Centers for Disease Control and Prevention. Wes1 and Wes2 are web surveys that were conducted by Westat using Blaise IS 4.7.

The server used for these tests had one Intel Xeon 1.3 Ghz processor and 1Gb of RAM, running Windows 2003 Server. WAS ran from a desktop workstation. In the tests reported below page delay is the average number of data fields on the interview pages times ten seconds. This is a substitute for changing the page delay in the script page by page to reflect the fields on the page. In the Wes1 and Wes2 datamodels large number of pages made it difficult to identify exactly what page in the script corresponds with a page in the IS application.

Table 2: Stress test of BRFS (60 minutes with an average page delay of 22.2 seconds)

Users	Pages	TPP*	Extra TPP
1	267	13.48	
25	6493	13.86	0.38
50	12687	14.19	0.70
100	24507	14.69	1.21
125	26565	16.94	3.46
150	29255	18.46	4.98
175	30284	20.80	7.32
200	31503	22.85	9.37
250	30429	29.58	16.09

*Time per page (TPP): $(\text{Min} * 60 * \text{Users}) / \text{Pages}$

As the number of users increase the performance declines, of course. But at what level of responsiveness (extra TPP) does it become a problem? Two seconds is used as the threshold. So for this survey with all functions (web, database, rules) running on one server of modest capacity (processors and RAM), the peak number of users that can be handled with an average time per page of less than two seconds greater than the single-user baseline is about 100 (the shaded cell).

For the much larger Wes1 data model, the threshold of two seconds of average extra time per page is reached between 40 and 45 users (Table 3). And for the second large data model, Wes2, the threshold is between 35 and 40 users (Table 4).

Table 3. Stress Test of Wes1 (60 minutes with an average page delay of 23.2 seconds)

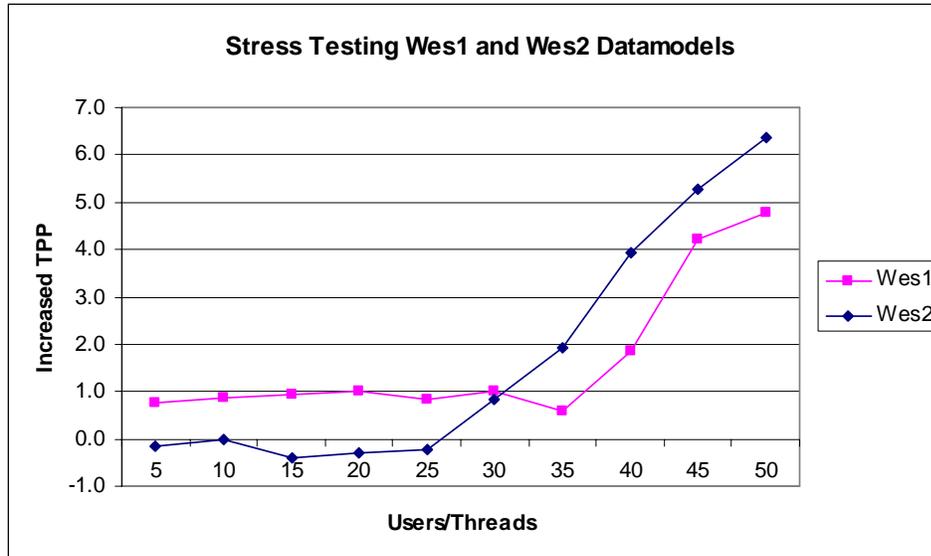
Users	Pages	TPP*	Extra TPP
1	203	17.73	
5	973	18.50	0.8
10	1936	18.60	0.9
15	2892	18.67	0.9
20	3842	18.74	1.0
25	4852	18.55	0.8
30	5764	18.74	1.0
35	6883	18.31	0.6
40	7350	19.59	1.9
45	7386	21.93	4.2
50	7990	22.53	4.8

*Time per page (TPP): (Min * 60 * Users) / Pages

Table 4. Stress Test of Wes2 (60 minutes with an average page delay of 11.8 seconds)

Users	Pages	TPP*	Extra TPP
1	356	10.11	
5	1808	9.96	-0.2
10	3570	10.08	0.0
15	5553	9.72	-0.4
20	7323	9.83	-0.3
25	9087	9.90	-0.2
30	9879	10.93	0.8
35	10482	12.02	1.9
40	10263	14.03	3.9
45	10536	15.38	5.3
50	10918	16.49	6.4

*Time per page (TPP): (Min * 60 * Users) / Pages



In examining the different test experiences of the Wes1 and Wes2 datamodels, the better performance of Wes2 may be explained by its lesser complexity—about 40% fewer fields and significantly less complex rules.

Follow-up activities are underway to enhance the server configuration, re-run the tests and examine the impact. A second IS server dedicated to running the rules will be added. The possibility is also being explored of using a more powerful server with more processors, speed, and RAM.

The Blaise team in the Blaise 4.8 Online Assistant reports in the section “Test results for web server performance” the significant impact of adding one or more dedicated rules servers. They used a large 1900-field datamodel with routing and checks. It peaked (< 2 seconds extra TPP) at 50 users with no dedicated rules server. Adding one rules server increased the peak to 100 users and adding a second increased the peak to 150 users.

3.2 An Ad Hoc Test

A colleague in looking at the test results questioned what an IS interview would look and feel like to a person when a WAS session was active. We tested this. Running the Wes2 script for 15 users, where the average extra TPP is .6 seconds, we independently conducted an interview on the application.

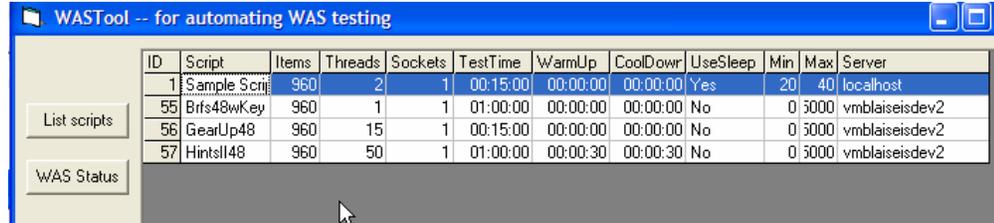
The performance was acceptable with most pages being returned by the server in a second or so. Occasionally the lag was greater, 3 or 4 seconds. An occasional delay of this duration felt normal by everyday web experience. Also, these delays seemed not inconsistent with the average delay time reported by WAS.

In this test we did encounter one functional anomaly. In all radio-button group items, the down-arrow key would not move the cursor in the list. One had to use the mouse to select. We reported this to the Blaise team.

This finding suggests that WAS might be valuable for usability testing to insure the application performs correctly when a number of other users are also active.

3.3 WAS and COM

A useful feature of WAS is that it has a COM interface, so one can automate it. We developed a VB application called WASTool to support running a series of tests. The main form displays the recorded scripts in the WAS database.

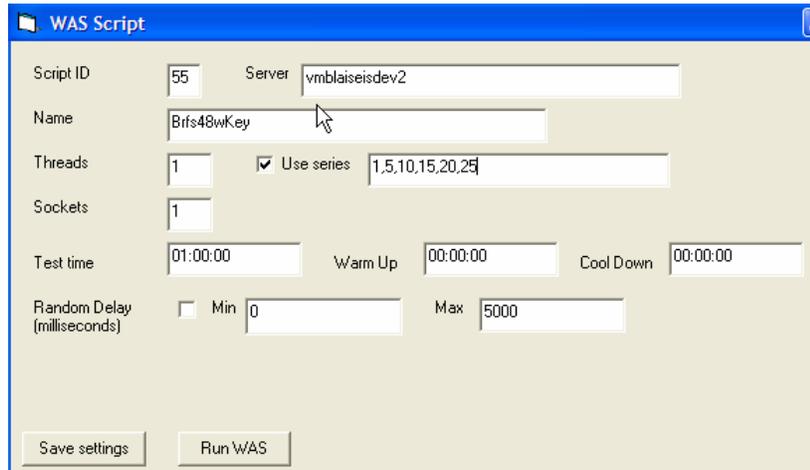


WASTool -- for automating WAS testing

ID	Script	Items	Threads	Sockets	TestTime	WarmUp	CoolDown	UseSleep	Min	Max	Server
1	Sample Scri	960	2	1	00:15:00	00:00:00	00:00:00	Yes	20	40	localhost
55	Brls48wKey	960	1	1	01:00:00	00:00:00	00:00:00	No	0	3000	vmblaiseisdev2
56	GearUp48	960	15	1	00:15:00	00:00:00	00:00:00	No	0	3000	vmblaiseisdev2
57	HintsI148	960	50	1	01:00:00	00:00:30	00:00:30	No	0	3000	vmblaiseisdev2

Buttons: List scripts, WAS Status

On clicking on a script, a form presents editable test parameters of the script—server, threads, test time, etc. The ‘Use series’ check box and the edit box next to it are added automation parameters.



WAS Script

Script ID: 55 Server: vmblaiseisdev2

Name: Brls48wKey

Threads: 1 Use series: 1,5,10,15,20,25

Sockets: 1

Test time: 01:00:00 Warm Up: 00:00:00 Cool Down: 00:00:00

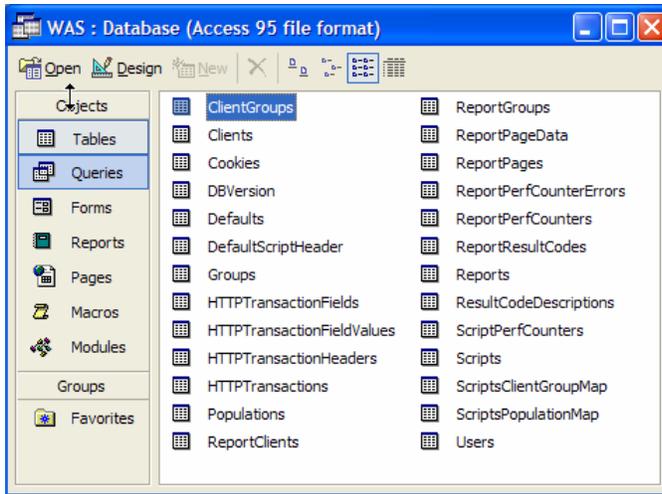
Random Delay (milliseconds) Min: 0 Max: 5000

Buttons: Save settings, Run WAS

If Use series is checked and the edit box has a comma-separated array of thread values, when Run WAS is pressed, WAS runs the script for each thread value in the series.

3.4 WAS Database

WAS uses a MS Access 95 database. The names of the tables in WAS.mdb suggest the breadth and detail that is recorded.



Other than changing a few script parameters in WASTool to support automated tests, we have only read or exported data from WAS.mdb using other systems such as SQL Server and Crystal Reports for reporting and other activities.

4. Other Metrics for IS Stress Testing

The Blaise team's stress testing method-- using WAS scripts with fixed page delays (10 seconds per response field on the page), one hour tests, and examining the time per page at different user levels-- is elegant and comparatively easy to implement.

We have done some work looking at Blaise interview records produced during WAS interview sessions as a supplement. After all, it is the interview data that we are ultimately concerned with. While we are not at a stage to present these statistics, it might be useful to discuss some of the techniques and issues.

To be able to identify test interview cases, it is important to have a case id primary key for the datamodel. However, WAS does not allow passing parameters to scripts. It runs the recorded script as is. Fortunately, in Blaise IS it is possible to adapt the interview starter page, biInterviewStarter.asp, so that when an interview is launched a unique server-generated session ID is set as the primary key for the interview case. The code is shown below.

```
Function CreateOptions(Hosts)
...
    If UseRunTimeOptions Then
...
        '*** setting sessionid as primary key
        Dim sCaseID
        sCaseID = Session.SessionID
        RootNode.setAttribute "KeyValue", sCaseID
        '***
...
End Function
```

Also, the IS journaling function must be implemented. With a journal enabled, a separate journal database is used and the IS server produces audit trail-type records during the interview process. When the interview starts and ends and when each page is sent to the user the journal records the date and time, primary key, server session id, and information on the action, mainly the page number and number of fields on the page.

The journal information allows one to compute interview times for cases in the IS application data file without having to add code to the data model. With this one can expand the information for each WAS test, adding to the average page time per thread level the number of completed Blaise interview and the statistics on the interview time: average, minimum, maximum, standard deviation.

As well, it may be possible to use the detailed page timings to learn other things about the process. This resembles the Windows Blaise audit trails, but it is much more limited because we know nothing about what the user did on any page, only when each page was sent from the server.

It seems possible, however, that analyzing the timing statistics by page for all cases in a study might be useful in identifying pages with different distributions of timing or outlier cases on key pages.

The steps being taken to test this approach are:

- Copy the application data and journal data files from the IS server.
- In Manipula export to ascii, appending fields on the application name and processing group.
- Import the ascii files into Sql Server tables
- Update the application and journal data records with the WAS.mdb report ID for the test based on the data time stamps and the WAS reports test start and end time

With interview data, journal, and WAS report results integrated it becomes possible to explore a variety of questions about the IS system, the application, and groups of interviews.

5. Conclusions

Stress testing of Blaise IS using the WAS system is a valuable approach to understanding the expected performance of Blaise IS for both specific applications and different server configurations. With major improvements in performance and scalability of Blaise IS in version 4.8, this testing process can demonstrate to external and internal clients whether the Blaise web survey system is ready to meet their needs. In addition to these top level issues, the testing process seems applicable to usability testing and using the journal information for process or paradata analysis.

Conversion of Blaise Databases to Relational Databases^{*}

Yogeeta Purohit & Latha Srinivasamohan, U.S. Census Bureau

1. Introduction

This paper discusses the conversion process from Blaise to relational databases using the OLEDB workshop. It is possible to convert Blaise databases to relational databases and vice-versa using the Blaise Component Package (BCP). This functionality was introduced in Blaise 4.5. The OLEDB (Object Linking and Embedding Database) workshop tool is an enhancement to this concept. It is used to create the BOI (Blaise OLEDB Interface) files that are used as a “Datalink” utility. The OLEDB tool is a graphical and user-friendly system. This tool was introduced in Blaise 4.7 and its latest builds have greatly enhanced it.

The authors of this paper researched the BOI file technology to determine whether they could simplify the back-end processing of the survey data. The paper will compare two methods of a coding system. The first is the currently implemented method that uses only Blaise software. The second method is a proposed process that converts the Blaise database to a relational database. Using the second method, the clients would be provided the output in the resulting relational database.

2. Survey Background

The American Time Use Survey (ATUS) was used as the sample survey for this research. This survey collects the time that a person spent on various activities in a 24-hour period. There is a limit of 90 activities reported per day. The interviewer can either pick the activity from the pre-coded list or record a new text entry, as shown in Figure 1. The interviewer has to record every activity during the day, so that every minute of a 24-hour period has been recorded for.

** Disclaimer: This document is to inform interested parties of ongoing research and to encourage discussion of Blaise instrument design issues. The views expressed are those of the authors and not necessarily those of the U.S. Census Bureau.*

Figure 1. Snippet of the activities row table

American Time Use Survey Webcati Instrument Ver 1.6.6 : October 2006 Production

Forms Answer Navigate Options Help Show Watch Window

Main Roster EDays FAQ S3 S4 S5 S8 S9 Exit

What did you do next?

- Read if necessary: An activity is anything you did during the day. Activities include both active tasks like socializing, preparing food, or eating; and more quiet tasks like thinking and relaxing. Right now, you are talking to me on the telephone. Talking on the telephone is one type of activity.
- Use the slash key (/) for recording separate/simultaneous activities.
- Do not use precodes for secondary activities.

1. Sleeping	8. Cleaning kitchen	30. Don't know/Can't remember
2. Grooming (self)	9. Laundry	31. Refusal/ None of your business
3. Watching TV	10. Grocery shopping	
4. Working at main job	11. Attending religious service	
5. Working at other job	12. Paying household bills	
6. Preparing meals or snacks	13. Caring for animals and pets	
7. Eating and drinking		

Pre-coded activities

	Start	ID	Activity	TIME	Hrs	Mins	Stop	Who	Who_2	Where	Where specify
[1]	4:00AM		Groomin	1		15	4:15AM				
[2]	4:15AM		Taking dog for a walk	1		15	4:30AM	0		1	Respondent's hom
[3]	4:30AM		watching TV	1		15	4:45AM	0		1	Respondent's hom
[4]	4:45AM		Exercising	1	1		5:45AM	0		1	Respondent's hom
[5]	5:45AM		Showering	1		30	6:15AM	0		1	Respondent's hom
[6]	6:15AM		Grooming	1		30	6:45AM				
[7]	6:45AM		Preparing meals and snacks	1		15	7:00AM	2		1	Respondent's hom
[8]	7:00AM		Driving to work	1		30	7:30AM	0		12	Car, truck, or moto
[9]	7:30AM		Checking emails	1		30	8:00AM	0		2	Respondent's wor
[10]	8:00AM		Working	1	2		10:00AM	0		2	Respondent's wor

3. Current Process (Blaise to Blaise)

Once the interviewer collects the Blaise data, the next step is to code the activities for analysis and drawing statistics. This second stage process of assigning generic codes is known as the coding system, which is described here. The current process uses only Blaise DEP and Manipula scripts to code the activities.

After the survey interviewing is completed, data from the activities table is extracted and a subset of the main instrument data is created. Once the subset is created, various scripts are run to extract and load the activity data into another instrument, the coding instrument. Here the activities are coded using generic codes provided by the sponsors. Figure 2 shows the flow of the process currently implemented.

The sponsors provide the coding rules and specifications for the coding system. Each activity code is subdivided in a three-tier layout. See Figure 3 for a sample view of the generic code split-up. During the coding process, the pre-coded activities are assigned their corresponding codes automatically, and the non-pre-coded activities (i.e., text entries) are coded using the coding system.

Figure 2. Standard process flow

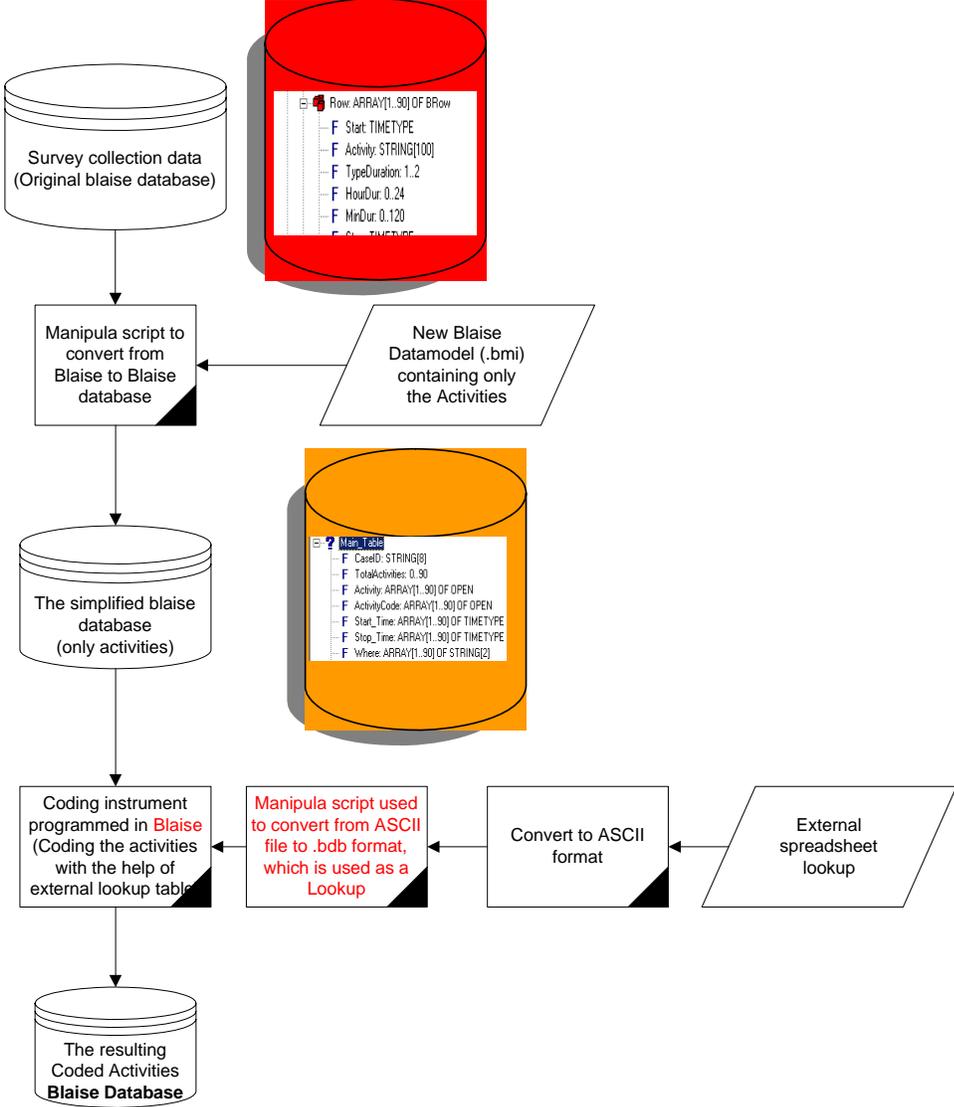
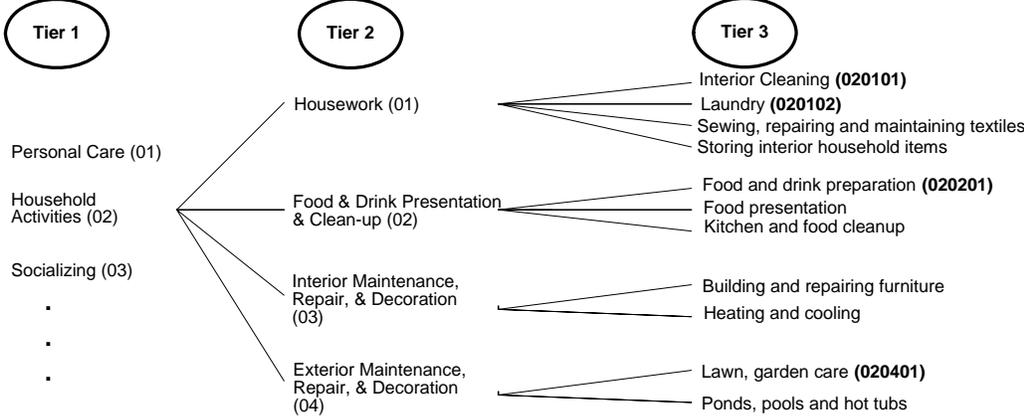


Figure 3. Three-tier data split-up of activities



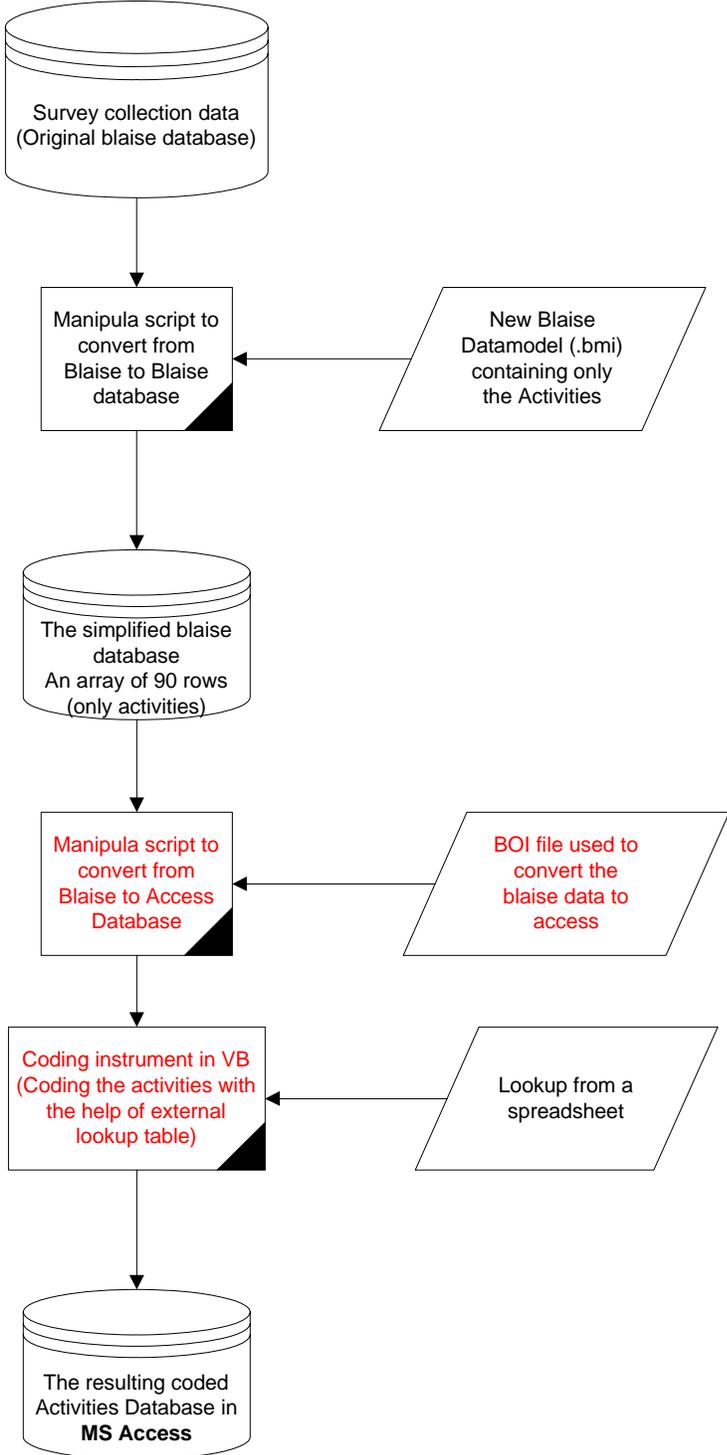
4. Alternative Process (Blaise to RDBMS)

Converting Blaise data into a relational database has been possible using earlier versions of Blaise, but it has not been simple. We will now look at the conversion of Blaise to a relational database management system (RDBMS) using the Datalink tool provided by Blaise 4.7.

The process copies the data from Blaise to a relational database using the BOI files. For our example, MS Access is the resulting relational database. The first step in the data conversion process is to create a subset of Blaise data. The reason for creating a subset is that in the original Blaise database the activities are embedded several layers deep within the blocks and tables¹. The new Blaise database consists of a block with 90-row arrayed activities, as shown in Figure 4. Once the data subset is created, the next step is to convert it to MS Access using the BOI file. This resulting Access database is used as the data storage for the coding system. A Visual Basic application is used for the coding system. Figure 4 explains the flow of the process:

¹ The intermediate Blaise db is not required. It was used to simplify the conversion task. The new developments to the BOI file allow complicated data extractions.

Figure 4. Flow of the alternative process



5. Relational Database Extraction Process

Blaise is an effective front-end survey application. However, the client may prefer data in relational database format for the back-end data processing. In this section, the paper addresses the steps to create a BOI file and customize it using the OLEDB package provided by Blaise 4.7. This helps convert data in relational databases such as MS Access, Oracle, MS SQL Server, and so on to Blaise and vice-versa. In our example, we convert data from Blaise to a MS Access database.

The first step is to create a BOI file. The OLEDB Toolbox (shown in Figure 5) is used to setup a BOI file based on the user requirements for the resulting relational database structure. The basic components needed to build the file are a .bdb (Blaise database) and its corresponding .bmi (Blaise data model).

1. Select the option to create the BOI file as shown in Figure 5. The OLEDB Toolbox wizard appears as shown in Figure 6.

Figure 5. OLEDB Toolbox

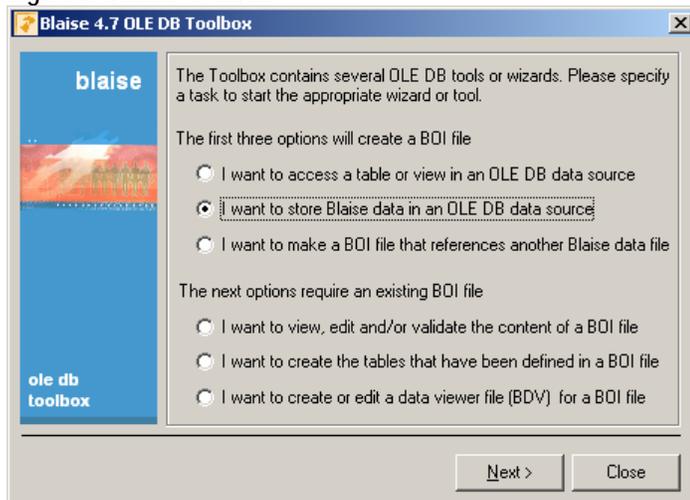
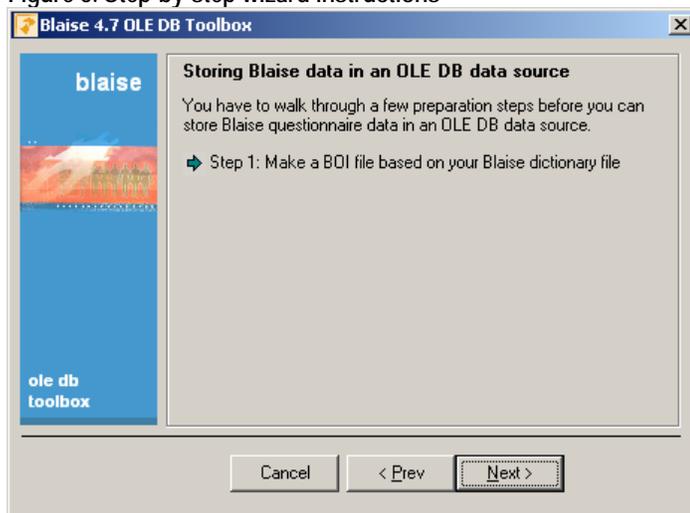
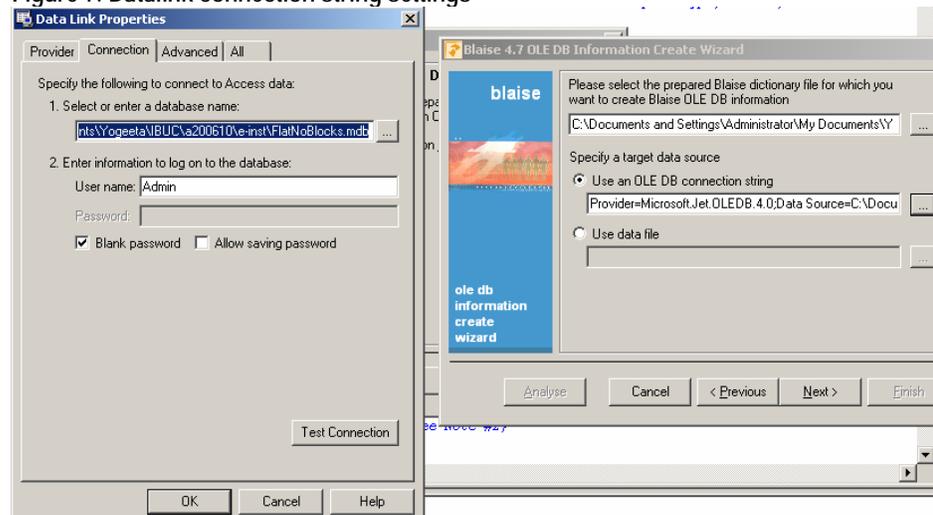


Figure 6. Step-by-step wizard instructions



- The next step is to pick the OLEDB connection string for the database to be converted. Here the user selects the OLEDB provider needed for the final database. For instance, we use the ‘Microsoft Jet OLEDB 4.0’ provider to connect to MS Access database. The properties for the relational database are set at this point, such as password protection, read/write mode, etc.

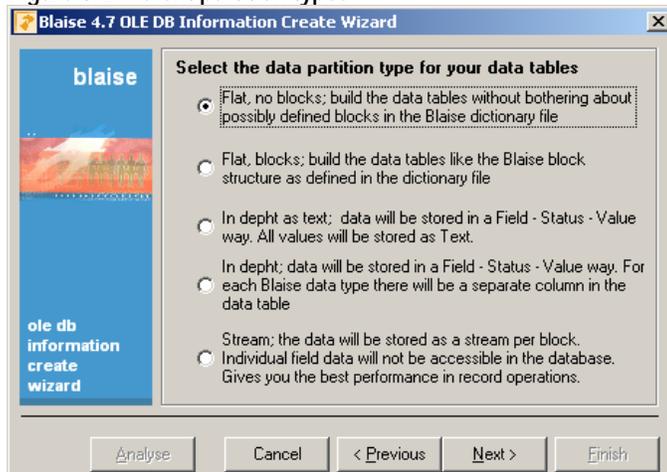
Figure 7. Datalink connection string settings



The OLEDB transfer provides several methods of dividing the data in the resulting database. The provider along with the BOI file helps setup the tables and the data connections. We selected the ‘Flat, no blocks’ as shown in Figure 8. This option transfers the data from a Blaise **field** to relational database **column** (Figure 10). MS Access 2000 has a limit of 255 columns per table. Therefore, once the limit is reached, a new table is automatically created and the remaining columns roll over to the new table. This process continues until all the fields from the Blaise database are transferred.

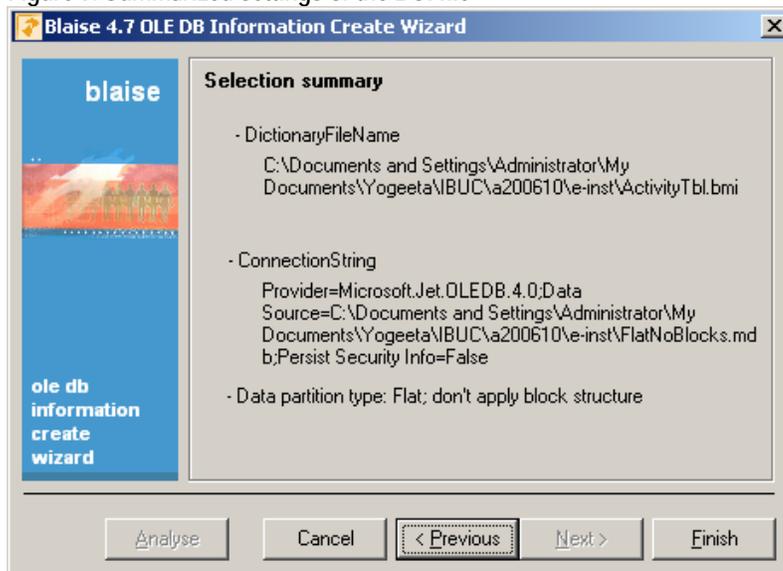
Since we did not have the need to set the secondary keys in the resulting database, we used the ‘Flat, no blocks’ option. The ‘In depth’ option is better suited for conversions that require secondary keys in the resulting database. As mentioned in Mr. Arno Rouschen’s abstract, “Generic Data Storage with Help from Blaise 4.8 Datalink”, primary and/or secondary keys are used to normalize the data, which leads to low database administration.

Figure 8. Different partition types



We have now created the BOI file, and a summary of its setting is shown in Figure 9.

Figure 9. Summarized settings of the BOI file



3. Once the BOI file is created, proceed to customize it using the 'Creating the table' wizard.

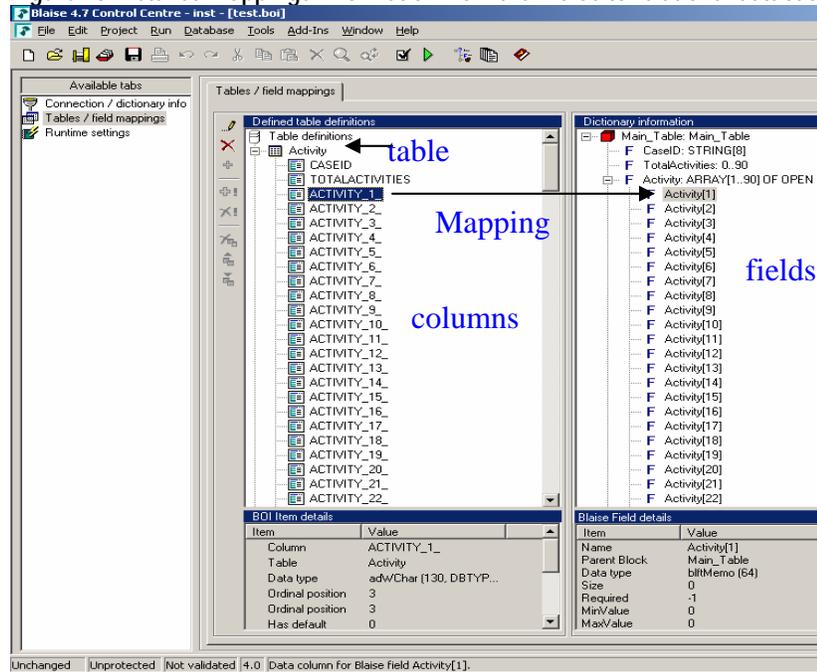
Note: The settings of the BOI tables can be either modified using the wizard or by using the OLEDB File Manager.

A few common examples in which a BOI file can be customized are given below:

- a. The connection string to the relational database can be reset.
- b. The connections between the Blaise fields and Access columns can be established. Figure 10 gives a good graphical view of this process. These are known as mappings. For setting up a better relational database

structure, the columns are moved around based on their relevancy to the tables. This improves the data accessibility by the back end application. Mappings can be added and deleted.

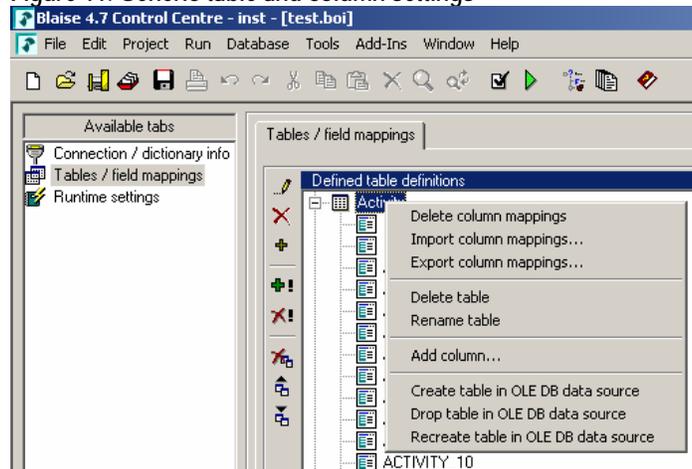
Figure 10. Detailed mappings information from the Blaise to relational database



- c. The settings of the tables can be updated in the BOI file. The list of different table functions is shown in Figure 11. Similarly, the column settings can be updated.

Note: The tables and the columns listed in the BOI need to be (re)created in the OLEDB. The BOI settings just provide an overview of the resulting database. The settings do not apply to the relational database until the tables are recreated. (Caution: Recreating the table wipes out the data in the corresponding current relational database table.)

Figure 11. Generic table and column settings



- d. The table and the column mappings shown in Figure 10 can be imported or exported to a file. A snippet of this file is shown in Figure 12. It is helpful to create a mapping file in case the table settings need to be reused or in case the BOI file gets corrupted.

Figure 12. Mapping file

```
[Activity]
CASEID=CaseID
TOTALACTIVITIES=TotalActivities
ACTIVITY_1_=Activity[1]
ACTIVITY_2_=Activity[2]
ACTIVITY_3_=Activity[3]
ACTIVITY_4_=Activity[4]
ACTIVITY_5_=Activity[5]
ACTIVITY_6_=Activity[6]
ACTIVITY_7_=Activity[7]
ACTIVITY_8_=Activity[8]
ACTIVITY_9_=Activity[9]
ACTIVITY_10_=Activity[10]
```

4. After creating the BOI file, the next step is to process it. A Manipula script is used to transfer the Blaise data to the Access database. The conversion in this case is record-oriented, which means it will convert the data record by record. This can be a time consuming conversion.

Note: In Blaise 4.7.1122 the BOI file can be customized for a set-oriented conversion. A drawback of this conversion method is that it converts databases consisting of only one data table. This approach is faster and efficient for copying data after the completion of data collection process. On the other hand, the record-oriented copy is helpful in a client-server environment since it locks only one record at a time. This issue needs further research.

Figure 13. Manipula conversion script

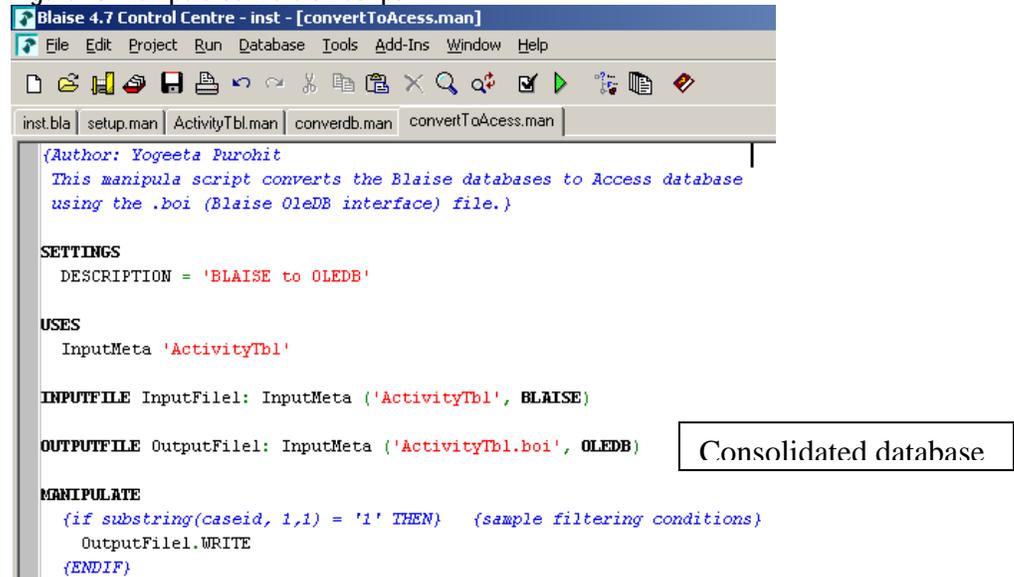
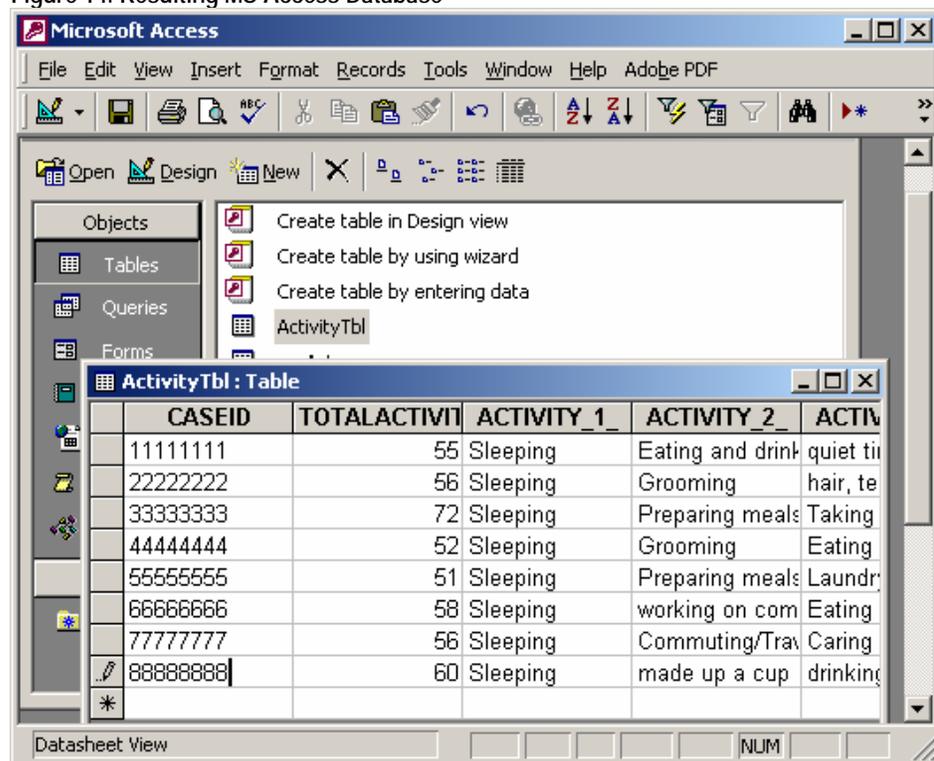


Figure 14 shows a snippet of the resulting MS Access database.

Figure 14. Resulting MS Access Database



- The resulting relational database is used by the second stage application for coding.

In our example, the Access database is used in a Visual Basic application. From this point onwards, the application requires no connection to the original Blaise

database. The Access database is updated with the user's entries in the application. This completes our coding system and the resulting output is stored in MS Access database.

6. Overview of Benefits and Drawbacks

This section identifies some benefits and drawbacks of the BOI files. These should be considered based on the client requirements.

6.1 Benefits

- The use of the BOI files allows the user to store data in a more normalized relational database.
- Better data administration may be provided using a true database.
- A subset of Blaise data could be shipped as well to protect sensitive data.
- BOI files are suited for backward compatibility. That is, the BOI functionality can be implemented on surveys built in Blaise versions 4.7, 4.6 and 4.5.
- BOI files do not require an additional programming language for the conversion.
- BOI files can be password protected, better in the client-server environment.
- The connectivity in the latest version of Blaise 4.7 has been improved.

6.2 Drawbacks

- Relational databases require more maintenance.
- All relational databases have a set limit on their number of columns and total record length. Therefore, it is essential to take into consideration these aspects while dividing the data in the BOI file.
- The BOI file adds an extra layer to the transformation as compared to the BCP, and therefore adds a delay to the data copy from Blaise to a relational database. (Refer to 'Listing Part 2 – Using Blaise 4.7 for Coding Listing Instruments' paper IBUC conference, 2006)

7. Technical Details

Blaise databases can be converted to relational databases provided there exists an OLEDB (Object Linking and Embedding Database) provider or ODBC (Open Database Connectivity) driver. This information is stored in the file called oledb.dbi.

An OLEDB provider is a software component that provides an interface for converting data to or from an OLEDB. Some of major OLEDB providers are Microsoft, Oracle and Sybase. Here is a list of the providers that are currently supported by Blaise:

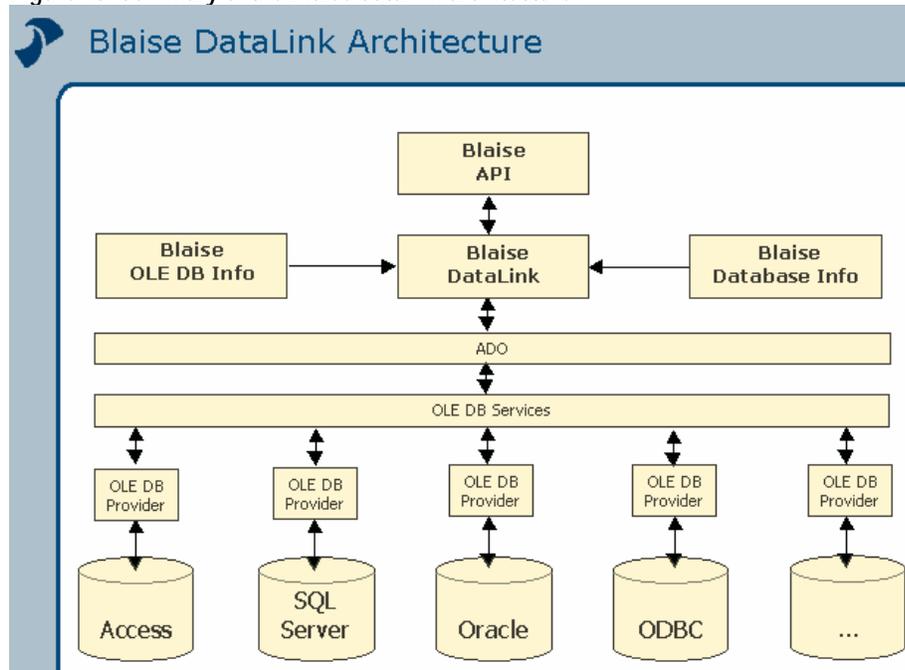
- Microsoft Jet 4.0 OLE DB Provider
- Microsoft OLE DB Provider for DB2
- Microsoft OLE DB Provider for ODBC
- Microsoft OLE DB Provider for Oracle
- Microsoft OLE DB Provider for SQL Server
- Oracle Provider for OLE DB
- SAS IOM Data Provider
- SAS/Share Data Provider
- Sybase Adaptive Server Anywhere Provider

An ODBC driver is an interface between an application and the DBMS (Database Management System), which translates the data to graphical format. The user easily understands this format. Some of the commonly used ODBC drivers that are currently supported by Blaise are:

- Microsoft Access Driver (*.mdb)
- Microsoft dBase Driver (*.dbf)
- Microsoft Excel Driver (*.xls)
- Microsoft ODBC Driver for Oracle
- INTERSOLV 3.10 32-BIT SequeLink
- Microsoft ODBC for Oracle
- MySQL
- MySQL ODBC 3.51 Driver
- Oracle ODBC Driver
- Oracle73
- SAS
- SQL Server
- SPSS 11.0.1 32-BIT Data Driver (*.sav)

Figure 15. Summary of the Blaise datalink architecture:

2



In order to use the BOI files, we need to have Blaise 4.7 work with the BCP 2. For that, the following files need to be registered:

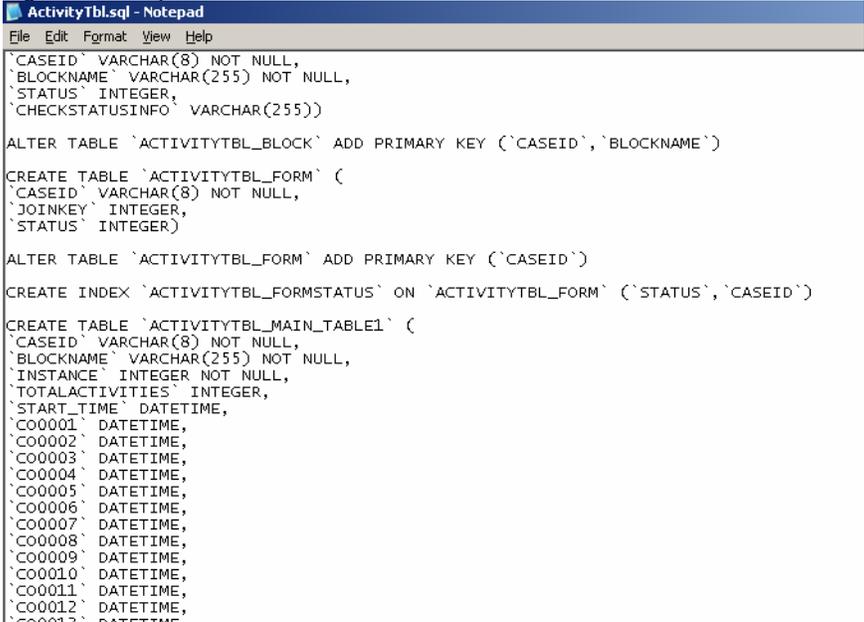
- **BlAPI4A2.dll** - Blaise API Objects Library
- **BlBOI4A2.dll** - Blaise OLE DB Interface Component
- **BlDBI4A2.dll** - Blaise Database Information Component
- **BlDL4A2.dll** - The Blaise DataLink Component (used for reading and writing from the OLEDB data sources)
- **SnReg1A.dll**
- **SiStrCIA.dll**
- **OLEDB.dbi** (Can not be registered, but must be in the same folder as BlDBI4A2.dll) – This file contains the OLEDB provider and ODBC driver information that are used with the Datalink. New relational database information can be added to this file in order for them to work with the BOI.

² Figure taken from the Blaise Reference Manual.

8. Some Useful Observations

1. The database software does not need to be installed as long as the OLEDB provider is available for data transfer. In our example, while creating the .mdb file, only the MDAC component package was installed, the MS Office software was not required. This is helpful in scenarios where the user does not need to look at the data in its original form, and can just have access through applications such as Visual Basic, C++, Java, etc.
2. The Datalink package can even work with non-relational data-files such as spreadsheets. A classic example of this is the conversion of an .xls file into a .bdb. The .bdb can be used for lookups and trigram searches in the Blaise instrument.
3. The “**create table wizard**” allows the user to create SQL scripts, which are used to set up the different relational databases. An example of this code is shown in Figure 16.

Figure 16. Sample SQL code:



```
ActivityTbl.sql - Notepad
File Edit Format View Help
`CASEID` VARCHAR(8) NOT NULL,
`BLOCKNAME` VARCHAR(255) NOT NULL,
`STATUS` INTEGER,
`CHECKSTATUSINFO` VARCHAR(255))
ALTER TABLE `ACTIVITYTBL_BLOCK` ADD PRIMARY KEY (`CASEID`,`BLOCKNAME`)
CREATE TABLE `ACTIVITYTBL_FORM` (
`CASEID` VARCHAR(8) NOT NULL,
`JOINKEY` INTEGER,
`STATUS` INTEGER)
ALTER TABLE `ACTIVITYTBL_FORM` ADD PRIMARY KEY (`CASEID`)
CREATE INDEX `ACTIVITYTBL_FORMSTATUS` ON `ACTIVITYTBL_FORM` (`STATUS`,`CASEID`)
CREATE TABLE `ACTIVITYTBL_MAIN_TABLE1` (
`CASEID` VARCHAR(8) NOT NULL,
`BLOCKNAME` VARCHAR(255) NOT NULL,
`INSTANCE` INTEGER NOT NULL,
`TOTALACTIVITIES` INTEGER,
`START_TIME` DATETIME,
`C00001` DATETIME,
`C00002` DATETIME,
`C00003` DATETIME,
`C00004` DATETIME,
`C00005` DATETIME,
`C00006` DATETIME,
`C00007` DATETIME,
`C00008` DATETIME,
`C00009` DATETIME,
`C00010` DATETIME,
`C00011` DATETIME,
`C00012` DATETIME,
`C00013` DATETIME,
```

9. Conclusion

We have reviewed two methods of creating a coding system. Both of these methods are efficient and can help different clients meet their requirements. The difference in the processes lies in the final output of the coding instrument. The alternative process provides data in a format that is easily accessible to the clients. The end product can be immediately put to use. Blaise Datalink has significantly simplified the task of transferring data from Blaise to RDBMS and vice-versa, which may help better serve the needs of clients.

10. Acknowledgments

We would like to acknowledge some people from the authoring staff, who provided valuable feedback on this paper. Our special thanks go to Karen Bagwell, Tom Spaulding and Rob Wallace for their valuable recommendations on the paper. Also, thanks to Michael Mangiapane, Roberto Picha, Ed Dyer and Leslie Fleet for their feedback.

11. References

Blaise 4.7 Reference Manual

Rouschen, A; Statistics Netherlands (2007): Generic Data Storage with Help from Blaise 4.8 Datalink, Abstract for the 11th International Blaise Users Conference 2007, Annapolis, MD, 2007.

Wallace, R., Picha, R., Mangiapane, M.; US Census Bureau, USA (2006): Listing Part 2 - Using Blaise 4.7 for Coding Listing Instruments, Statistics Netherlands, Presented at the 10th International Blaise Users Conference 2006, Netherlands, 2006.

Wikipedia.org. OLE DB and OLE DB Providers.
16 Jul. 2007 <http://en.wikipedia.org/wiki/OLE_DB>.

Webopedia.com. ODBC.
16 Jul. 2007 <<http://www.webopedia.com/TERM/O/ODBC.html>>.

Deploying a C# .NET Object onto a Laptop to be used with a Blaise 4.7 Data Collection Instrument

Mécène Désormice, Daniel Moshinsky, Thomas Melaney, U.S. Census Bureau

1. Background

The U.S. Census Bureau is currently designing and developing an improved annual survey, the re-engineered Survey of Income and Program Participation (re-engineered SIPP). The survey is intended to provide statistical information on household income and participation in income assistance programs and serve as an improvement to the Survey of Income and Program Participation (SIPP). In order to facilitate accurate recall for re-engineered SIPP, chronological data for the survey will be collected with the help of an Event History Calendar (EHC) which, as the name implies, will ask respondents to record key events into a calendar. The functionality of the EHC is being designed and developed in a C# .Net environment producing a DLL which will be called from the Blaise DEP.

This document will identify the steps necessary to deploy a Blaise survey with a .NET DLL component onto a system that lacks the .Net component framework. An alternative successful strategy will also be discussed. Implications for deploying this kind of instrument onto hundreds of laptops configured for use by field interviewers will be discussed as well.

The initial development of the C# DLL was done at the U.S. Census Bureau using a standard Personal Computer (PC) that was supported within a network environment. The initial programming, testing and revisions as well as the presentations for review and design consultation with the program specification specialists all took place at a programmer's PC station.

Within a few weeks the development had reached a stage when it was appropriate to move the application to a laptop computer. The actual data collection operation for the survey is to be a Computer-Assisted Personal Interview (CAPI) operation. It was important to verify early within the development that the screen design was appropriate and that the program would operate well on the type of laptop to be used by the field interviewers, referred to as field representatives (FRs). It was also important to have the instrument operating on a laptop as a convenient way to demonstrate the new instrument to interested stakeholders in different locations. The laptop configuration currently used by the U.S. Census Bureau for the field data collection operation is the Dell Latitude D400 with a Windows 2000 operating system.

After initially porting the EHC DLL to the laptop, the programmers discovered that the program could not be accessed. It was necessary to engage in a variety of modifications to resolve the porting issues one step at a time. Searching the internet for articles with information about the proper methodology for the deployment was helpful but frustrating in that there were few articles that examined the entire process of transferring a C# DLL to a station that did not include the MS Visual Studio software. Many times, suggested approaches were inappropriate for the situation at the U.S. Census Bureau. The laptops to be used for data collection have very restricted access by the end-users (field representatives) and the security on those laptops must be extraordinarily tight. The

questions in many of the surveys at the U.S. Census Bureau ask respondents for information that could, at the questionnaire level, identify individuals and provide information about their families, households and economic status that would be considered highly sensitive. In accordance with the U.S. Census Bureau's privacy principles as stated under Title 13 of the U.S. Code, the U.S. Census Bureau takes measures to assure that privacy is strictly preserved and that any data or tabulations that are made available to clients or the public are free from any personally identifiable information.

Within this document, the project is called DEWSEHC. This name is used within the coding and reflects an earlier reference name used for the re-engineered SIPP Event History Calendar.

Please note that this report is meant to inform interested parties of ongoing research and to encourage discussion of these technical issues. The views expressed in this document are those of the authors and not necessarily those of the U.S. Census Bureau.

2. Purpose

The Event History Calendar (EHC) is a Dynamic Link Library (DLL) that is written in Visual Studio 2005. It is intended to be loaded on the FR's laptop and launched from the Blaise Instrument through an Alien Router call.

The purpose of this document is:

- Show how to add and use the Blaise API Components package in C#.
- Show how to deploy and register a C# DLL or component on the FR's laptop or on a PC other than the development machine (PC)
- Identify the files that need to be sent to laptop or PC along with DLL.

3. Adding the Blaise Components Package (BCP) to a C# project

To add the BCP, you need to first create a new Visual Studio project using the Windows Control library. In this case, we are going to use the DEWSEHC project as the example for this exercise. Below is a picture of the Solution of the project (see Fig. 1).

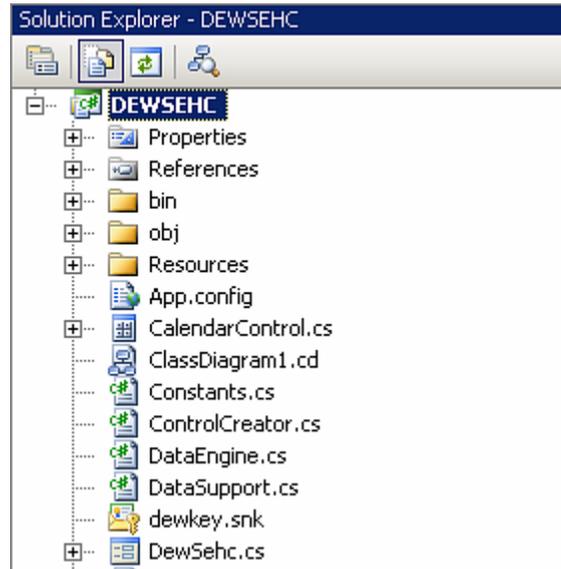


Figure 1.

Next a reference to the Blaise API component must be added. This can be done by selecting Add Reference from the project menu or by right-clicking the References folder in Solution Explorer (see Fig. 2).

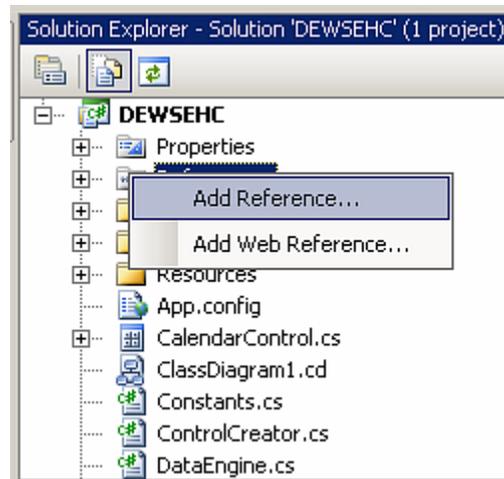


Figure 2.

Select the COM tab from the Add Reference Window, and then select Blaise 4.7 API Component (see Fig. 3).

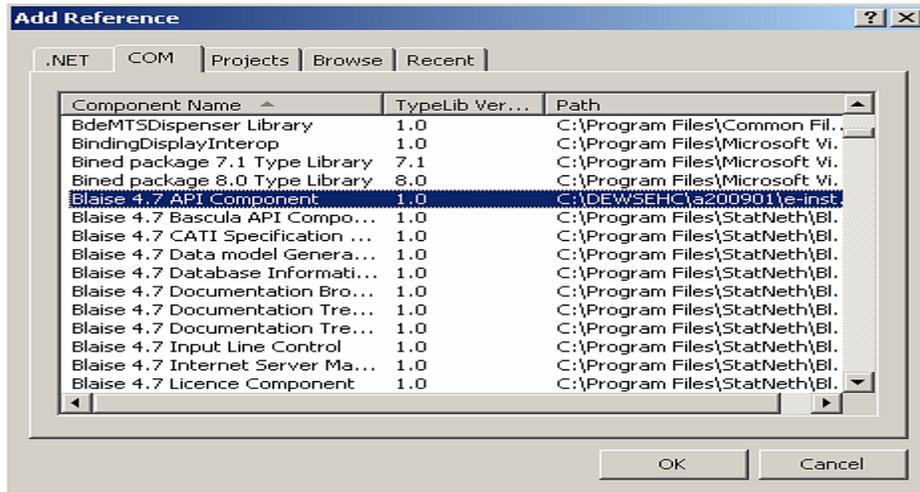


Figure 3.

C# will automatically add BI-API4A2.dll to the project and create two new files: Interop.BI-API4A2.dll and Interop.SiCstrCIA.dll (see Figure 4).

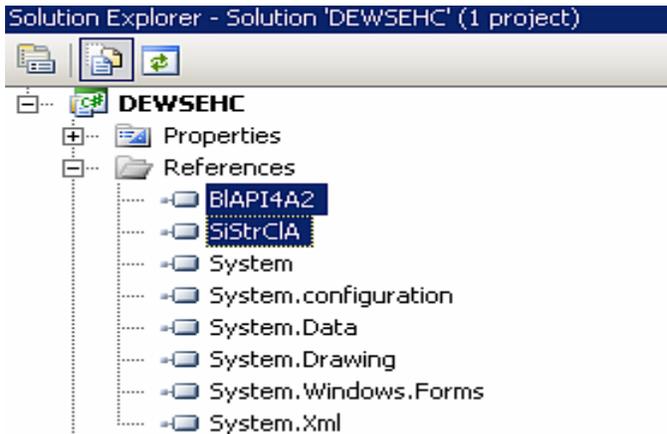


Figure 4.

These files are placed in several locations under the project folder. For example, these files may be found within the Debug folder.

Then, the reference to BIAPI4A2 must be included in the source code of the project's UserControl class. The keyword "using" is used in C# to include references into the application. From within the Blaise DEP, the Alien Router call will invoke the GetMeAField method, which, in turn, calls the EHC form – as shown below.

```
using System;
using System.Runtime.InteropServices;
using BIAPI4A2;
namespace DEWSEHC
{
    public partial class CalendarControl : UserControl
    {
        private BIAPI4A2.Database db;
        private BIAPI4A2.DepState ds;

        public CalendarControl(){InitializeComponent();}
        private void CalendarControl_Load(object sender, EventArgs e){}
        public void GetMeAField(BIAPI4A2.Database DB,
                                BIAPI4A2.DepState DS)
        {
            DEWSEHC.DewSehc mCalendar = new DEWSEHC.DewSehc(DB, DS);
            mCalendar.ShowDialog();
        }
    }
}
```

4. Preparing for deployment

The deployment of DEWSEHC to the laptop requires the following additional steps:

1. Verify that Windows Service pack version 3 or higher is installed on the laptop.
2. The .NET framework distributed package (dotnetfx.exe) must be loaded on the laptop and installed. The dotnetfx.exe is located under
C:\Program Files\Microsoft Visual Studio 8\SDK\v2.0\BootStrapper\Packages.
3. Create the .NET object Strong Name. This can be done by right-clicking the project folder in the project Solution Explorer and selecting Properties. Then, select the Signing tab, check “Sign the assembly”, and select “New” from the Dropdown box (see figures 5 and 6).

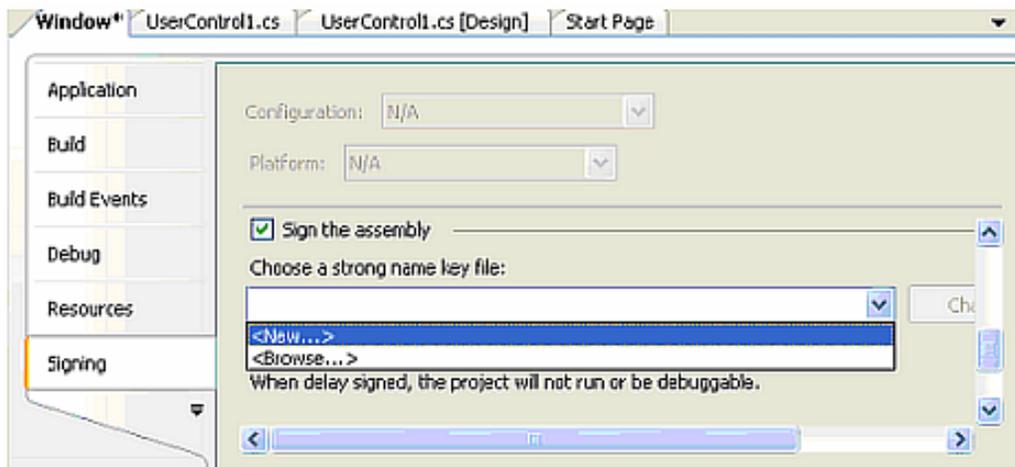


Figure 5.



Figure 6.

- Set ComVisible to true. To set ComVisible to true:
 1. Right-Click on the project name, then select Properties
 2. On the Properties Window, Click on Build and Check Register for COM interop.
 3. The dropdown box, select “**On**”

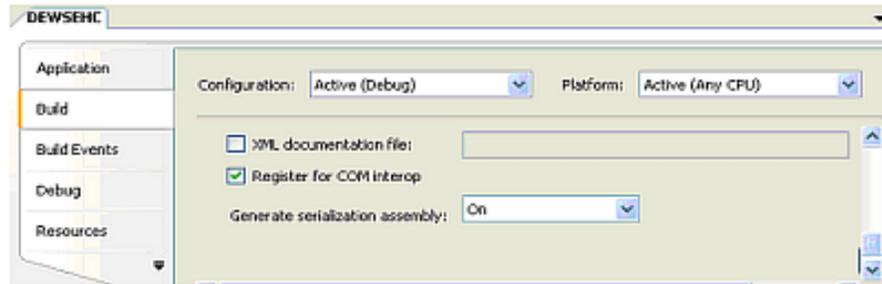


Figure 7.

- Rebuild the application.

5. Deploying the DLL

Now that the DLL has been prepared, we are ready to deploy. The following files must be copied into the project directory on the laptop:

- BI-API4A2.dll – This is the run-time file that will interact with the Blaise DEP
- Interop.BI-API4A2.dll – This file provides information needed to call methods exported from the Blaise API shared library
- Interop.SiStrCIA.dll – This file also provides information needed to call methods exported from the Blaise API shared library
- DEWSEHC.dll – This is the application DLL
- DEWSEHC.tlb – This is the application type library automatically generated at build-time

Register the application DLL using the following command.

```
C:\WINNT\Microsoft.NET\Framework\v2.0.50727\regAsm.exe /codebase
/tlb=DEWSEHC.tlb DEWSEHC.dll
```

Register the Blaise API DLL as follows:

```
Regsvr32 BI-API4a2.dll
```

A batch file that includes these two commands can be created.

Now, the Blaise instrument should successfully invoke the DLL on the laptop.

6. Alternate Suggested Strategy

In communications with colleagues at Westat and Statistics Netherlands, an alternative strategy was suggested for resolving the configuration problem.

The suggestion was to create a setup program to deploy the .NET DLL by using the following steps.

1. Give the .NET DLL a Strong Name
2. Create a “setup and deployment project” within Visual Studio
3. Run the setup on the target machine and install the files in an appropriate directory.

This approach was not appropriate for our situation. The “setup” program requires that the user input values to reply to questions. Our current process of distributing our applications to thousands of laptops in the field is fully automated. It did not seem to be practical to alter our current production routines to run the setup program since we could avoid the issue by using the strategy described in this paper instead.

7. Questions for Further Investigation

The project discussed in this paper is still in a relatively early stage of development. We have yet to attempt to deploy the data collection instrument to field laptops on a large scale. We will be working to determine what, if any, adjustments may need to be made to our data collection instrument or to our survey distribution system in order to assure a smooth interface.

Designing an e-Form to Collect Survey Data

David Kinnear, Office for National Statistics, UK

1. Introduction

The Office for National Statistics (ONS) currently uses Blaise as a collection tool for its social surveys, via telephone, face-to-face and self-completion. It has also used web collection for the internal ONS Staff Perception Survey. However, the introduction of Blaise IS with version 4.7 and BASIL with version 4.8, however, allows ONS the possibility of collecting business survey data over the internet. At present, ONS uses TDE (Telephone Data Entry) and imaging software to capture data from paper forms. Using Blaise Internet will provide ONS the opportunity to offer businesses another option for returning their data.

This paper provides details of the current method used by ONS to collect data for its business surveys and will show the results of our project to design an e-Form based on the existing ONS forms. The poster session at the conference will demonstrate the versions of this e-Form created using Blaise IS and BASIL. I will be available to highlight issues encountered whilst creating the e-Forms, and show images of the completed forms that were uploaded to the ONS intranet site.

2. Current business survey data collection methods used by ONS

ONS uses 2 methods for the collection of business survey data.

2.1 Method 1 – Paper forms

Survey samples are generated from the ONS Inter-Departmental Business Register (IDBR). Sample information includes, amongst other things, the name of the business, contact person, IDBR reference number and classificatory data. This information is then combined with template survey forms to produce paper versions which are then dispatched by post. Completed forms are then returned, and the data is imaged and scanned. The scanned data is held in electronic files which are then transferred overnight to the survey database system, which is used to calculate the survey results.

2.2 Method 2: Telephone Data Entry (TDE)

Although the process for creating the TDE forms follows that described for paper forms, there are several other considerations:

- the maximum number of data items suitable for collection via TDE is considered to be 9
- only numeric data and comments are suitable for collection via TDE
- a unique identifier generated by the collection database will be used to avoid mis-keying of reference numbers

- the data will be subject to limited inquiry specific validation at the TDE source
- survey forms, letters and other documentation must comply with the standard for TDE forms
- the inquiry dialogue must comply with the standard for TDE dialogues

ONS applies strict control over TDE dialogue and forms. They must be formally signed off by survey representatives, the Forms Processing Centre and Methodology Group prior to transfer to TDE. The whole process is overseen by the Data Collection Initiatives team.

The TDE process is as follows:

Stage 1 - Introduction/Goodbye/Transfer

- Welcome
- Change to dialogue – highlight changes to previous versions
- Contributor ID – contributor keys in ID number as printed on survey form
- Incompatibility – contributor telephone not recognised by TDE system.
- Alternative options provided eg transfer to ONS operator or survey contact
- Contributor ID accepted
- Data success – response recorded successfully
- Sign-off message.

Stage 2 – Confirmation codes

- Describe the action before the code e.g. to confirm press 1
- Data entry values and comments are followed by the hash or gate key
- Data entry values are played back for confirmation
- Confirm values e.g. to confirm press 1; to re-enter press 2

Stage 3 – Dates

- Confirm whether contributor is responding for the current period or a previous period
- Confirm response is for the standard period (set by ONS) or calendar period
- If non-standard period, the contributor keys 2 digits for the first day, last day and month of reporting period. If unsuccessful at recording dates, the contributor has 2 further attempts before being asked to contact the ONS

Stage 4 – General data

- If Euro flag is set for the contributor the dialogue will replace pounds with euros.
- Financial questions are asked in "thousands of pounds"
- All data responses (except nils) should end with a hash

Stage 5 – Comments

- Any failure in validation will mean:
 - first asking the contributor to confirm the data
 - then asking for an explanation – either by leaving a message or by transfer to desk (depending on survey)
- Change in data message will give the contributor a choice of leaving a message or transferring to desk (in office hours)
- Contributor offered the option of leaving contact name and number

3. Developing an e-Form

3.1 Background

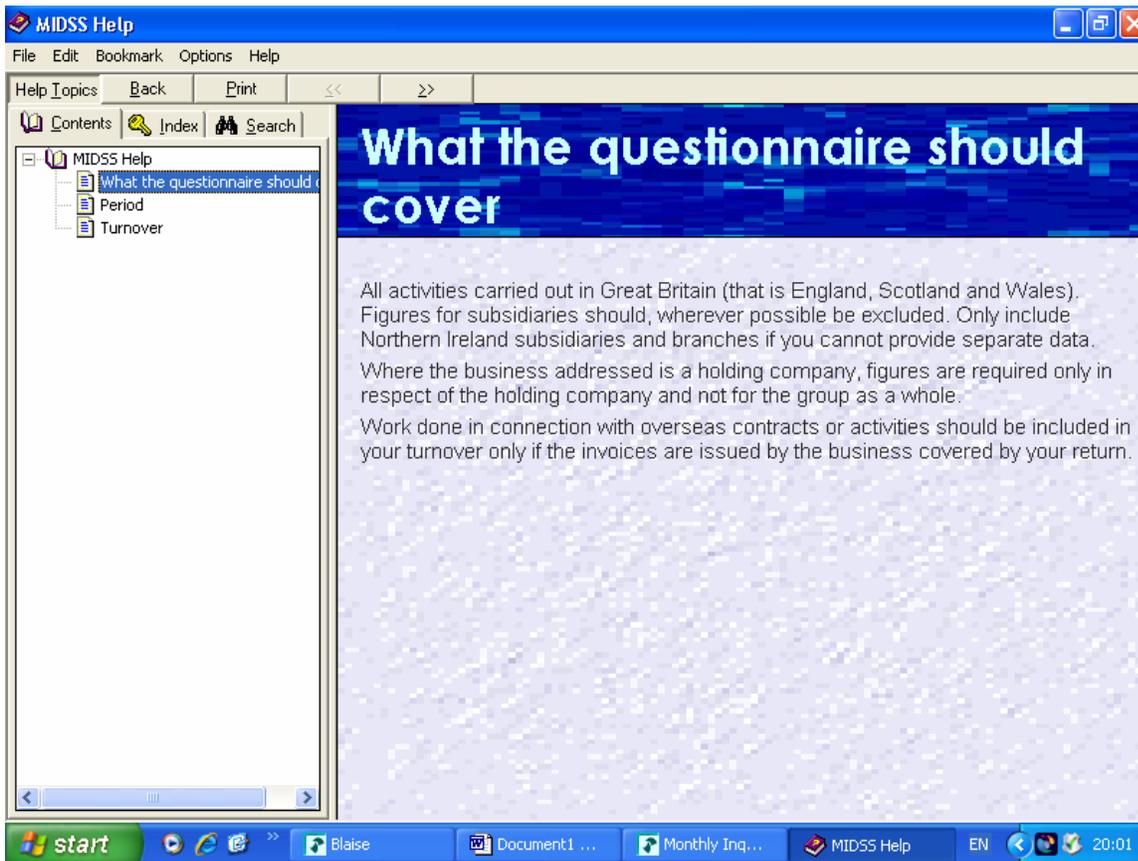
Internet collection offers potential efficiency savings over Telephone Data Entry and paper collection of data. The current process involves considerable resources from the reprographics unit and output handling to produce the paper forms and then dispatch them via the postal system

The Blaise Development Standards and Support team has been eager to extend the use of Blaise to ONS's Business Surveys, and has been actively engaged in this over the past three years. Using Blaise 4.8, BASIL now gives us the opportunity to demonstrate recent progress and how Blaise is a real contender for these surveys.

Whilst trying to replicate the paper form electronically we wanted to add features and functionality that would aid the contributor in completing the form, and ultimately result in the collection of better quality data. Using the extra flexibility that an e-Form allows, we have adapted the form in various ways:

- break down questions into clearly defined sections
- clear navigation between the defined sections of the e-Form
- extension of Q-by-Q Help [Setchfield, 2006] to provide an on-line help guide. This package allows for easy navigation around the help screens, and provides a more professional look. ONS has been using this help package for some time on our social surveys, and has met with a favourable response from interviewers. The Robohelp settings use the same standards as our social surveys (see Figure 2)
- on-line checking and validation. This will place more of the responsibility for data consistency on the contributor, and the use of error messages will aid this
- direct electronic links to the person in ONS responsible for the questionnaire. This should encourage greater interaction between the office and contributors

Figure 2 – Screenshot of help facility for the survey



3.2 Questionnaire structure and standards

3.2.1 Signing In

An 8-digit reference number will be emailed separately to the survey contributor. This will be used to access the rest of the questionnaire (see Figure 3).

Figure 3 – Survey Sign In screen

Monthly Inquiry into Distribution and Service Sectors

national STATISTICS

Sign In

Please enter your 8 digit reference number

Any queries ?

Email:
david.kinnear@ons.gsi.gov.uk

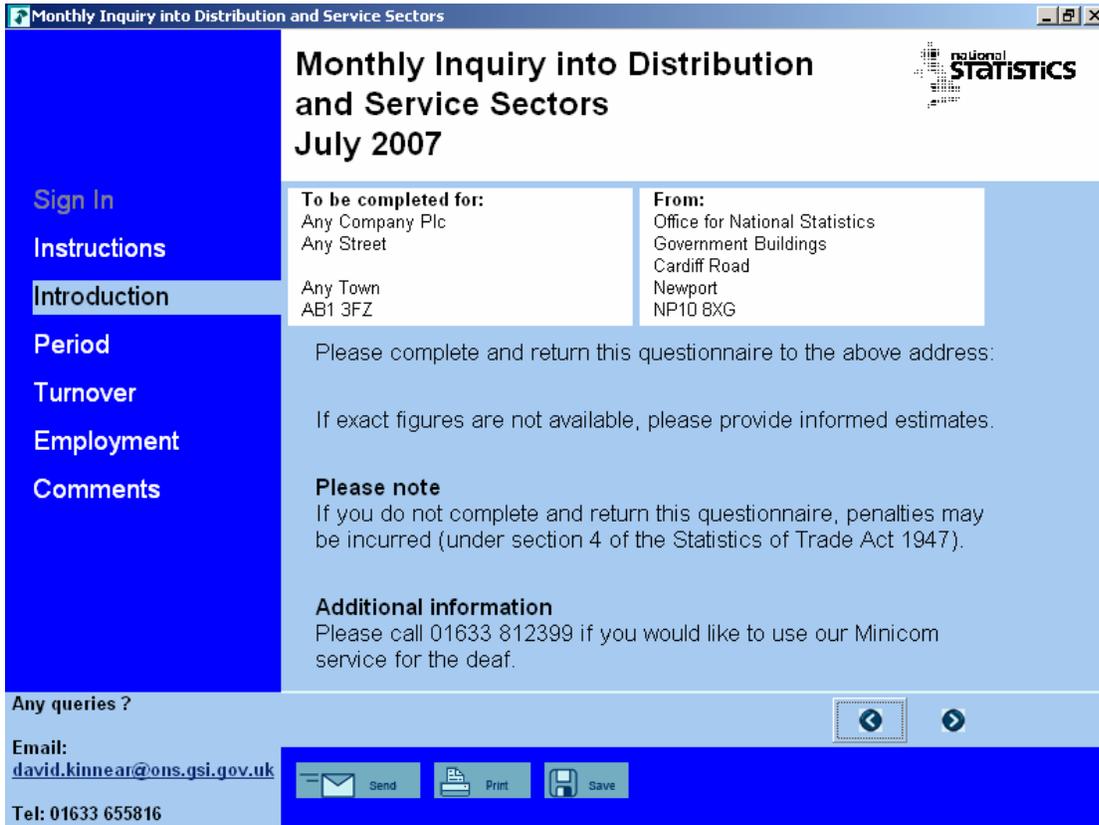
Tel: 01633 655816

Send Print Save

3.2.2 Creating individual e-Forms

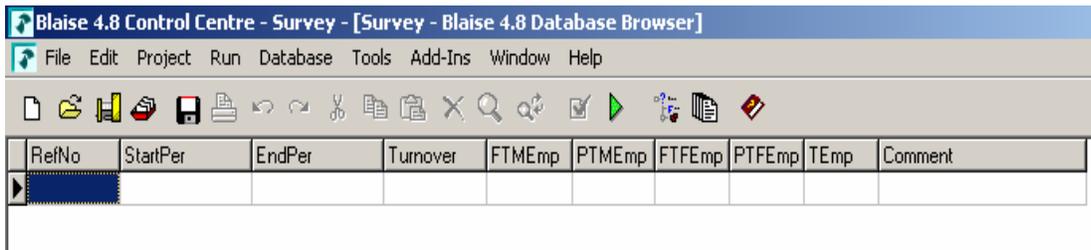
Every survey period, sample files are created from the Inter-Departmental Business Register (IDBR). Using Manipula, information such as the IDBR reference number, company name, address and contact name will be separated from this main file and stored in a separate ASCII file. This will be e-mailed with the form, to allow its contents to be displayed on the Introduction screen (see Figure 4). This will ensure that the data return is for the correct reporting unit.

Figure 4 – Survey Introduction screen



Although the company name and contact details will be displayed on the questionnaire, for added security these won't be included in the completed Blaise questionnaire database file (see Figure 5).

Figure 5 – Questionnaire database file



3.2.3 Screen standards and appearance

ONS has had considerable experience in developing standards for social survey questionnaires. However, this was our first attempt at creating a business survey form.

In designing the screen layout, we wanted to include the following:

- Distinct areas which would be consistent throughout the questionnaire. This will encourage familiarity with what to expect on each screen
- A distinct sign-in page for the contributor to enter their access reference number. Once this has been successfully input, the contributor can access the rest of the questionnaire. The sign-in page then becomes read only, and the access number cannot be re-entered
- Clear navigation and structure. Contributors will need free navigation around the questionnaire, to enable them to return to questions that have been skipped, or amend previously entered data. The questionnaire map displayed in the left pane will give the contributor an idea of progress through the form (with the current page always highlighted), and allow them to jump between sections
- Easy access to help pages by inserting a link alongside each question. The prominent display is to encourage use of the help facilities
- ONS contact email and phone number displayed at all times. The email address will link directly to the mail client
- 800 x 600 pixels screen resolution to provide clearer viewing (as in our social surveys)
- Minimise the use of scroll bars, reducing the risk of a contributor missing an instruction

Figure 6 – Screenshot of survey form

Monthly Inquiry into Distribution and Service Sectors

Monthly Inquiry into Distribution and Service Sectors
July 2007

national STATISTICS

Sign In
Instructions
Introduction
Period
Turnover
Employment
Male Employees
Female Employees
Total Employees
Comments

Male employees
Full Time
Part Time

Female employees
Full Time
Part Time

Total all employees

Any queries ?

Email:
david.kinnear@ons.gsi.gov.uk

Tel: 01633 655816

Send Print Save

A comments page is also included, to allow the contributor to record explanations for any unusual reported values, and other details such as change of contact name or address details.

Options allow the contributor to print their return, save at any point or send the questionnaire back to the office. When selecting the 'send' button, a warning will be displayed if any of the questions have not been completed.

4. Next steps

The Blaise code to create the questionnaire has been completed. We now need to consider other issues, such as:

- Data security – we will involve our IT specialists to consider online security issues, transmitting confidential data via e-mail. However, the ONS has successfully conducted a confidential Staff Perception Survey, using Blaise IS, on the ONS intranet
- Performance issues. We will also be analysing the performance when emailing a quantity of forms at the same time, as several ONS surveys have a substantial sample size

- Integration into existing ONS systems. Files containing individual address and contact details will need to be generated from the existing IDBR system and incorporated into the Blaise questionnaire. Completed questionnaire data then needs to be inserted into existing results processing systems. Manipula seems the logical choice to do this
- Introducing this method of data collection as an alternative to existing methods, or as the sole method. We will need to react to comments and suggestions by contributors, and incorporate valid requests where necessary

5. References

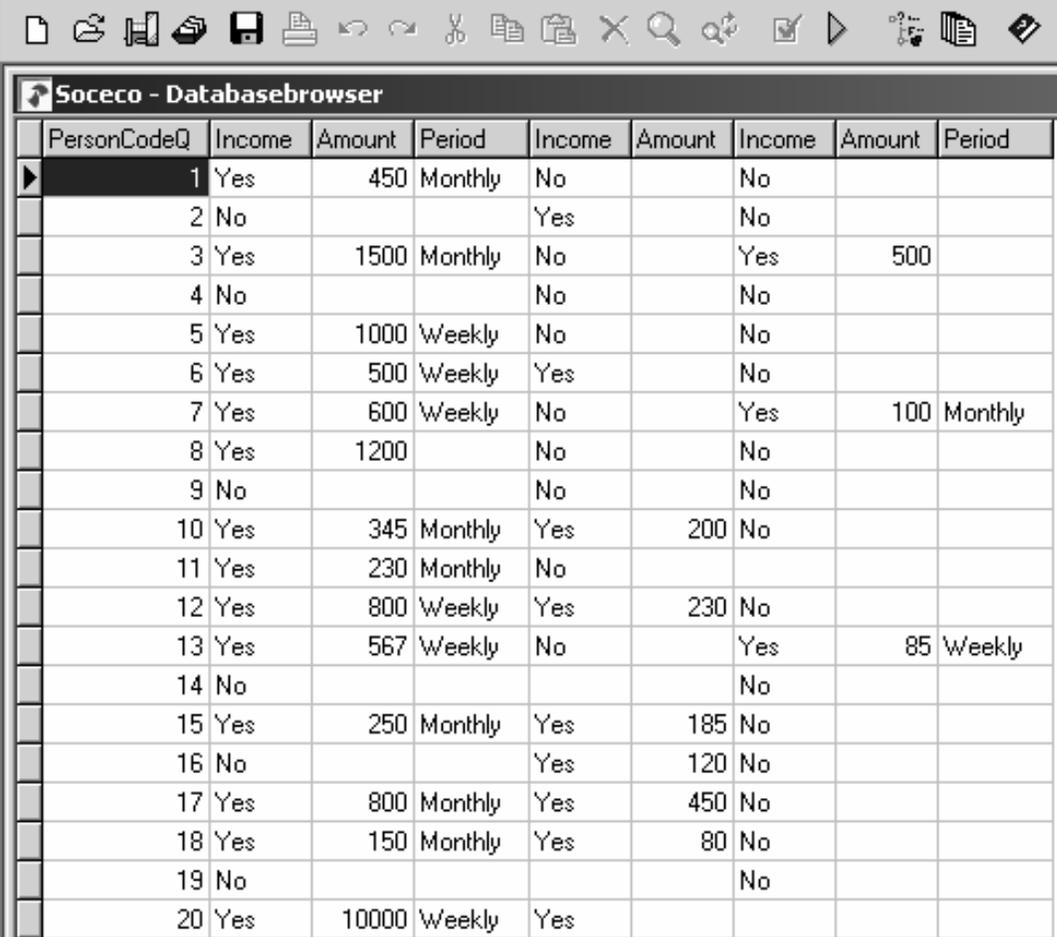
Setchfield, C., 2006, *Training and Learning Opportunities using Blaise*, IBUC proceedings 2006

Miscellaneous Examples of Programming in Blaise and Manipula

Rob Groeneveld, Statistics Netherlands

1. Anonymizing a Blaise database

Starting with a Blaise database with identifiable records, we would like to get rid of the primary key which may be some identifying variable like the name of a company or a registration code for a person or company. It is desirable not only to strip the identifier, but also to sort the records in the database in random order so as to nullify the association between the order of the records and the individuals. As an example, we take the records in the example database Soceco.bdb, part of the examples of the Blaise Component Pack (in StatNeth\Blaise 4.7 Enterprise\Samples\BCP). Here is the complete database:



PersonCodeQ	Income	Amount	Period	Income	Amount	Income	Amount	Period
1	Yes	450	Monthly	No		No		
2	No			Yes		No		
3	Yes	1500	Monthly	No		Yes	500	
4	No			No		No		
5	Yes	1000	Weekly	No		No		
6	Yes	500	Weekly	Yes		No		
7	Yes	600	Weekly	No		Yes	100	Monthly
8	Yes	1200		No		No		
9	No			No		No		
10	Yes	345	Monthly	Yes	200	No		
11	Yes	230	Monthly	No				
12	Yes	800	Weekly	Yes	230	No		
13	Yes	567	Weekly	No		Yes	85	Weekly
14	No					No		
15	Yes	250	Monthly	Yes	185	No		
16	No			Yes	120	No		
17	Yes	800	Monthly	Yes	450	No		
18	Yes	150	Monthly	Yes	80	No		
19	No					No		
20	Yes	10000	Weekly	Yes				

The identifying field is PersonCodeQ, which is in the primary key block:

```

DATAMODEL SocEco
PRIMARY PersonCode
BLOCK PersonCodeBl
  FIELDS
    PersonCodeQ "What is your personal code?" : 1..50
  RULES
    PersonCodeQ
ENDBLOCK
FIELDS
  PersonCode: PersonCodeBl
TABLE TIncome "Income specification"
(etc.)

```

As a first step, we replace the primary key field by a secondary key:

```

DATAMODEL AnonSocEco
SECONDARY
  RK = PersonCode
BLOCK PersonCodeBl
  FIELDS
    RandomKey : 0..200
  RULES
    RandomKey
ENDBLOCK
FIELDS
  PersonCode: PersonCodeBl
TABLE TIncome "Income specification".
.
.

```

This datamodel does not have a primary key. Note that the range of the secondary key is somewhat larger than the range of the original primary key. We will fill this field with random numbers and it is advantageous to have some extra space.

```
PersonCode.RandomKey := RANDOM(200)
```

A first Manipula setup transforms the data from the model SocEco to the model AnonSocEco, leaving out the primary key but adding the random secondary key field RandomKey. There is no association between the value of the secondary key and the previous primary key because the secondary key is assigned randomly.

Why can't the new key be the primary key? The method we have chosen does not preclude the possibility of duplicate values for the new key; hence it cannot be a primary key. If we would use a procedure to generate random unique keys the new key could be the primary key. However, generating new unique keys is more difficult and takes more time per record.

Why is the range for the secondary key bigger than the original range? Because it lessens the chance of getting duplicate keys.

Now we want to put the records in the order of the secondary key to cancel the association between the order of the records in the original Blaise file and the order in the resulting Blaise file. You cannot simply say

```

{ wrong! }
SORT
  PersonCode.RandomKey

```

because Blaise files cannot be sorted. We will process the records in the order of the secondary key and write them to a new Blaise database in this order. An additional advantage is that the datamodel for the new Blaise database can have features such as having the secondary key as the primary key or omitting the secondary key altogether.

The essential part of the second Manipula setup is:

```
INPUTFILE
  In1: InputMeta ('SocecoRandomSort', BLAISE)
  SETTINGS
    KEY = SECONDARY(RK)
OUTPUTFILE
  Out1: OutputMeta('AnonSoceco', BLAISE)
```

We tie everything together in a Manipulus program:

```
{ File: RandomSortRecs.man }
PROCESS RandomSortRecs
  { This setup starts with the Blaise file with the field
  PersonCode as the primary key, adds a random number as a
  secondary key while omitting the primary key, then processes
  these records in the order of the secondary key writing the
  records to a new Blaise file }
  SETTINGS
    INPUTPATH = ''
  USES
    OutputMeta 'AnonSoceco'
  OUTPUTFILE
    Out1: OutputMeta('SocecoRandomSort', BLAISE)
  AUXFIELDS
    Reslt: INTEGER
  MANIPULATE
    Reslt := CALL('RandomSort1 /OSocecoRandomSort')
    Reslt := CALL('RandomSort2')
    Out1.ERASE
```

Note that the intermediary database SocecoRandomSort.bdb is deleted at the end of this program.

2. Printing Open Fields In Field Order In A Blaise Database

In a Blaise database obtained as a result of a survey, the task is to print the Open Fields in field order, i.e., firstly the answers from all records in the first open field, then the answers from all records in the next open field, and so on. That is, if we have three records with three open fields, like this:

OpenField1	OpenField2	OpenField3
aaa	bbb	ccc
ddd	eee	fff
ggg	hhh	iii

we want output in the form

```

OpenField1
=====
aaa
ddd
ggg

OpenField2
=====
bbb
eee
hhh

OpenField3
=====
ccc
fff
iii

```

For the output file, we use this datamodel:

```

DATAMODEL OpenQuestions
FIELDS
  Aline: STRING[1200]
ENDMODEL

```

And use the output file type PRINT.

The start of the Manipula program goes like this:

```

SETTINGS
  CONNECT = NO
  AUTOREAD = NO

USES
  InputMeta 'AllQuestions'
  OutputMeta 'OpenQuestions'

INPUTFILE
  InputFile1: InputMeta ('AllQuestions', BLAISE)
OUTPUTFILE
  OutputFile1: OutputMeta ('OpenQuestions.txt', PRINT)

```

The setting `CONNECT = NO` is used because there are no fields in input and output which have the same names. The setting `AUTOREAD = NO` is used because we will be traversing the input file from start to finish once for each open field, under programmatic control. The datamodel `AllQuestions` contains both the normal fields and the open fields.

To write all the answers to an open field `OpenField1` to output, we can use the following fragment:

```

Inputfile1.RESET
InputFile1.READNEXT
OutputFile1.LINE(2)
OutputFile1.PRINTSTRING('OpenField1 ')
OutputFile1.PRINTSTRING(FILL(' ', 80))

```

```
REPEAT
  OutputFile1.ALine := OpenField1
  IF LEN(ALine) > 0 THEN
    OutputFile1.WRITE
  ENDIF
InputFile1.READNEXT
UNTIL InputFile1.EOF
```

This produces the output text in the form

```
OpenField1
=====
aaa
ddd
ggg
```

Note that the field is mentioned twice, first its name as the string 'OpenField1' and the second time as itself in the assignment:

```
ALine := OpenField1
```

When we want to turn this sequence of statements into a procedure, we face the problem that parameters in a procedure cannot be of OPEN type. The only types allowed are: user-defined, INTEGER, REAL, STRING, DATATYPE or TIMETYPE. A user-defined type may not be based on a BLOCK. So this is not possible:

```
{ wrong! }
PROCEDURE OutputAField
  IMPORT strFieldName: STRING
  IMPORT FileName: OPEN
INSTRUCTIONS
  Inputfile1.RESET
  InputFile1.READNEXT
  OutputFile1.LINE(2)
  OutputFile1.PRINTSTRING(strFieldName)
  OutputFile1.PRINTSTRING(FILL(' ', 80))
  REPEAT
    OutputFile1.ALine := FileName
    IF LEN(ALine) > 0 THEN
      OutputFile1.WRITE
    ENDIF
    InputFile1.READNEXT
  UNTIL InputFile1.EOF
ENDPROCEDURE
```

Hence, we must remove the assignment with the OPEN type from the procedure:

```
OutputFile1.ALine := FileName
```

And create two procedures. Since the loop REPEAT...UNTIL cannot be split across two procedures, we make a procedure for the heading:

```
PROCEDURE Heading
  PARAMETERS
    IMPORT Fieldname: STRING
  INSTRUCTIONS
    Inputfile1.RESET
    InputFile1.READNEXT
    OutputFile1.LINE(2)
    OutputFile1.PRINTSTRING(Fieldname)
    OutputFile1.PRINTSTRING(FILL(' ', 80))
ENDPROCEDURE
```

And a procedure for writing to the output file:

```
PROCEDURE WriteToOutput
  INSTRUCTIONS
    IF LEN(ALine) > 0 THEN
      OutputFile1.WRITE
    ENDIF
    InputFile1.READNEXT
  ENDPROCEDURE
```

For every OPEN field we use the two procedures plus the assignment and the REPEAT loop:

```
Heading('OpenField1')
REPEAT
  OutputFile1.ALine := OpenField1
  WriteToOutput
UNTIL InputFile1.EOF
Heading('OpenField2')
REPEAT
  OutputFile1.ALine := OpenField2
  WriteToOutput
UNTIL InputFile1.EOF
.
.
```

This produces the desired output of the OPEN fields. Note that only fields which have some content are written because we test if LEN(ALine) is greater than zero.

3. Partially Filled, Nested Arrays In A Survey

The usual way of programming an array with a varying number of filled elements in a Blaise survey is to ask in the first place for the number of repetitions, e.g., the number of people in the household, followed by a FOR loop to put a set of questions to each member of the household, for example,

```
..
  FIELDS
    HHSize "How many people are there in this
household?": 1..10
    Person: ARRAY[1..10] OF BPerson
  RULES
    HHSize
    FOR i := 1 TO HHSize DO
      Person[i]
    ENDDO
```

Another way is asking, after each set of questions, if there is a further set of questions to be asked. An example is this datamodel:

```
DATAMODEL Sandwiches
BLOCK Blk_Sandwich
  FIELDS
    Name "What sandwich did you eat?": STRING[15]
```

```

        MoreSandwiches "Did you eat more sandwiches?": (Yes,
No), EMPTY
    RULES
        Name
        MoreSandwiches
    ENDBLOCK { Blk_Sandwich }
    LOCALS
        I: INTEGER
    FIELDS
        Sandwich: ARRAY[1..10] OF Blk_Sandwich
    RULES
        Sandwich[1]
        FOR I := 2 TO 10 DO
            IF Sandwich[I - 1].MoreSandwiches = Yes THEN
                Sandwich[I]
            ENDIF
        ENDDO
    ENDMODEL

```

After each question about the sandwich you ate the question is asked if you ate more sandwiches. If the answer is Yes, the question about the next sandwich is asked. Note the way the FOR loop works: the first element must be asked before the FOR loop is started, because inside the FOR loop reference is made to the question MoreSandwiches in the previous element of the array.

We can expand this model by asking about the meals the person ate today. Each meal can include sandwiches or not:

```

DATAMODEL Meals
BLOCK Blk_Sandwich
    FIELDS
        Name "What sandwich did you eat?": STRING[15]
        MoreSandwiches "Did you eat more sandwiches?": (Yes,
No), EMPTY
    RULES
        Name
        MoreSandwiches
    ENDBLOCK { Blk_Sandwich }
BLOCK Blk_Meal
    FIELDS
        Name "What meal did you eat with sandwiches?":
STRING[15]
        Sandwich: ARRAY[1..10] OF Blk_Sandwich
        MoreMeals "Did you eat more meals today with
sandwiches?": (Yes, No)
    LOCALS
        I: INTEGER
    RULES
        Name
        Sandwich[1]
        FOR I := 2 TO 10 DO
            IF Sandwich[I - 1].MoreSandwiches = Yes THEN
                Sandwich[I]
            ENDIF
        ENDDO

```

```

    MoreMeals
ENDBLOCK { BMeal }
FIELDS
    Name "What is your name?": STRING[15]
    Meal: ARRAY[1..10] OF Blk_MEal
LOCALS
    I: INTEGER
RULES
    Name
    Meal[1]
    FOR I := 2 TO 10 DO
        IF Meal[I - 1].MoreMeals = Yes THEN
            Meal[I]
        ENDIF
    ENDDO
ENDMODEL

```

The field MoreMeals asks if any more meals with sandwiches were eaten.

A datamodel like this can be applied to an inventory of the knowledge of people in a department, both in terms of the areas of expertise and the length of the time spent in specific jobs. A few example questions and answers:

First level:

Question: In which divisions did you work?
 Answer: Social Statistics

Second level:

Question: In which areas within the Division of Social Statistics did you work?
 First answer: Household Surveys
 Second answer: Population Statistics

Third level, elaborating first answer:

Question: In which areas within Household Surveys do you have experience?
 First answer: Blaise Datamodelling – Question: How many years of experience do you have with Blaise Datamodelling? – Answer: 1 to 4 years
 Second answer: Analysis – Q: How many years? – A: More than 10 years

Third level, elaborating second answer:

Question: In which areas within Population Statistics do you have experience?
 Answer: Sample design – Q: How many years? – A: 5 to 9 years.

A varying number of answers can be given at any level.

New CAI-operation at Statistics Norway

Hilde Degerdal and Jan Haslund, Statistics Norway

1. Background

1.1 Our recent CAI operation

In 1999 Statistics Norway built a new system for Computer Assisted Interviewing out by the field interviewers. The system has been a good operation for our work. However, after a while you find things that have to be improved.

As a system for field interviewers, the main weakness of our system is the communication solution. It was built as a dial-back RAS connection with a reverse proxy server, polled from inside. Firewalls and encryption are also in use. It is a rather complicated solution, but it had to be this way to meet the requirements of the Data Inspectorate and The Protector of Privacy in Norway. As a consequence of the complexity, the rate of unsuccessful connect attempt is too high.

In our system each respondent is sent to an interviewer as a package. If the interviewer for one reason or another is unable to perform the interview, he has to connect again and return it to the central database. Then, in the office, the staff has to find a new interviewer for the respondent and prepare a new transfer. This work is quite time consuming.

Another problem is that when the respondent packages are out with the interviewers we have little control of the collection progress by each interviewer.

1.2 CATI

Another issue is CATI. In 1991 we started a small CATI-operation in Statistics Norway. However, when we provided our field interviewers with laptops in 1995, the central CATI-staff was no longer required. The field interviewers had capacity enough to carry out those interviews as well. It is also convenient to “fill up” the work load for interviewers who have too few hours just conducting face to face interviews.

That was also the situation when we built our recent CAI in 1999. But after a short period of time, in 2000, it was decided to build up a central CATI-staff again. It was a requirement that CATI should be incorporated in the CAI-system in such a way that it should be possible to shuffle respondents between the two systems. What we did then was to consider CATI as one interviewer in the CAI-system. So each day a large amount of respondents are shuffled to and from the “CATI-interviewer”. This is not the best solution, but for a small scale CATI- operation it has worked.

Last year the CATI-operation expanded to about 40 workstations. This meant that it was impossible to have the entire amount of respondents being shuffled within the CAI-system. So today we have two independent systems, and we are not satisfied with that.

2. The project

A project is going on to make a new system. It is called the SIV –project. SIV = System for Interview Activities. The project’s aim is to make a system that meets the requirements described below.

2.1 Requirements for the new system

A main issue is that the respondents shall not be distributed out to the laptops. We want them to be stored and kept in one common database. This means that the field interviewers need to work online when they conduct their interviews.

The system must offer possibilities for different levels of dedication of respondents to interviewers. In some surveys, or phases in the data collection, it may be that we want all the interviewers to work in a traditional CATI-manner. In another situation it may be more suitable to have dedicated respondents to interviewers, or groups of interviewers.

Surveys with mixed or multimode data collection design will be more and more frequent. Therefore, we want to build a system that can handle data coming in from different sources (interview, web, paper) in a simultaneous way.

The importance of a stable and safe communication solution cannot be stressed too much. A system as described, with online interviewing, sets high requirements to up-time.

We want to develop a system built up of modules, so that it can be built piece by piece. This will also make it possible just to renew some of the pieces of the system in the future. Another requirement is to have expertise on the whole system in-house. The system must be built with focus on user-friendliness, and the whole process shall be documented in a user-friendly manner throughout.

2.2 Organization of the project

There are a lot of issues to deal with in the project. We have divided the tasks into five subprojects.

1. **Matters related to the interviewers.** This subproject shall develop strategies on levels of dedication of respondents to interviewers. They are going to decide on matters concerning the interviewers working conditions, such as a payment system. Another issue for the group is to evaluate software, hardware and access that can be useful for the interviewers in their work. The last main task for this group is to make good guides for the interviewers on registration of a contact with a respondent. It is important that all interviewers follow the same rules for registration since they are going to handle the same respondents.
2. **Workflow in the survey projects.** We want more standard methods and more streamlined workflow in our work. This subproject shall describe routines; make templates and guides that will form a framework for that. They will give input to the developers on data that are needed to support the data collection process.
3. **Development of the core of SIV.** This group is going to design and develop the central database and operation in the office, the interfaces both in the office and for the interviewers. Their work is going to be done on the basis of requirements from subprojects 1 and 2.
4. **Communication system.** This subproject shall decide and develop solutions for communication, telephony and hardware.

5. **Training of the users.** This subproject will develop a training module for the interviewers and for the staff in the office on the new system. They are also responsible for the quality of the documentation of the entire system.

In addition to the subproject groups we have a group of interviewers and a group from the central staff as reference groups. The steering group is the Department of IT and Data Collection management team, supplemented by two representatives from the labour unions.

2.3 Schedule

We aim to have a new operation in production by 1 April 2008.

3. Ideas so far (July 2007)

3.1 Issues to consider

An important part of the project is of course to make a stable communication solution of good quality. We plan to equip the field interviewers with broadband. In order to reduce the costs, we are also considering the use of IP-phones. That also means that the lines have to be of good quality.

The work on designing the solution has started, there are some ideas, but the main work on this will be done during late autumn or winter.

An issue we have focused on is the system for face to face interview. As mentioned, we want an online solution with the respondent kept in the central database. However, we have done some investigation and concluded that the coverage on wireless connection throughout the whole country is not good enough. So we can't make a solution that requires online interviewing also for face to face interviews. Therefore we will need to build functionality for downloading questionnaires and respondents to the interviewers' laptops, and subsequent synchronizing of the collected data back in the database when the interview has been conducted. We want to have control of all the respondents' status all the time, so we want the appointments to be made online, also for the face to face interviews. That will also make it easier for us to ask another interviewer to do the interview if the first interviewer is unable to do it.

The change from individual respondents to a shared list of respondents for telephone interviews will be the most noticeable change for the interviewers. This will make some challenges. The most important one, as for all CATI-organizations, is how to match the capacity of interview work hours to the actual workload. We think however, that our challenge will be even worse, since the interviewers are not in-house. We will obviously need a monitoring tool for the supervisors. The supervisor's work will require a possibility to direct the interviewers into a survey that needs them. Our idea is to make some parameters for the system that will offer a ranked list of surveys, which is presented to the interviewers when they log on to the system. The list should be read like this: green surveys means "go on", yellow will be "if you want, you may choose this one", red ones have already too many interviewers logged on to them, and will not be possible to start. In addition the supervisor may need a possibility to send "blips" out to the interviewers to advise them to switch to another survey, when that is needed.

Another worry is whether the interviewers' sense of responsibility for making good appointments and persuasive arguments will be the same, when they have common

respondents, compared to the situation of today, where they have their own respondents. Some of the field interviewers have big concerns about this issue. Our experience for telephone interviews in present operations is that CATI does not have a significantly lower response rate than field interviews.

3.2 Screens

For the **interviewers** we first want a screen that offers:

- A list of CATI-surveys with common respondent lists. The colour of the printed name of the survey shall indicate which of them they should select.
- A list of their own appointments for today
- Buttons to:
 - Start a selected CATI -questionnaire
 - Transfer respondents for face to face interview
 - Synchronize data from a conducted face to face interview
 - Check if more respondents for face to face interview are available in the neighbourhood. This may be useful if one interviewer has idle capacity, and another interviewer in the area is absent.
 - Open a list of their respondents for face to face or telephone
 - Open a list of their appointments. Own appointments are mainly appointments with their respondents for face to face interviews. Perhaps it should also be possible to transfer CATI-respondents to this list. This functionality may be useful e.g. if the respondent wants to be interviewed at an odd time, which is ok by the specific interviewer.
 - Look up a respondent in the database by name, id, or phone number. It should also be possible to start the interview with the actual respondent.
 - Reports. Some reports will be available, that will give the opportunity to compare own results, and time use, with the average ones.
 - Report work hours
 - Report travel log
 - Questionnaire testing. It would be useful to use interviewers with idle capacity from the entire interviewer staff to test questionnaires. Today a limited team, living near the office, mostly does this job.
 - Read survey manuals. Maybe the survey can't be selected from the list by the interviewer before he has read the manual.
 - Open a questionnaire for training. Maybe the survey can't be selected from the list by the interviewer before he has trained on the questionnaire.

- Access to available web questionnaires. The interviewers should know which surveys have a web questionnaire available, so they can offer the respondent the option to fill this in. It may also be useful for the interviewer to access a certain web questionnaire to help a respondent filling in a questionnaire.

In the interview interface, when calling the respondent the interviewer shall have access to as much information about the respondent and earlier contact (-attempts) as possible.

One of the first things the interviewer has to ask the respondent about is if he has read the letter to respondents. If he hasn't the interviewer can read it for him, or send him a new letter. It must be possible from the system to send a new letter as an attached pdf to an email, or just to click a button, and a message will be sent to the office, telling which letter has to be send to whom for sending paper letters.

For the **supervisors**, we want a screen with a list of ongoing surveys with number of available respondents in daybatch, appointments, and number of interviewers logged on to it. A window within the screen will continuously show messages coming in from the interviewers.

The screen must also have buttons to:

- Get an overview over appointments spread in time
- Give higher/lower priority to a survey if it has too few, or too many interviewers logged on to it
- Change CATI parameters for a survey
- Inspect daybatch for a survey
- Make new daybatch for a survey
- Send a message to an interviewer or a group of interviewers, e.g. to switch survey
- Stop interviewing on a survey
- Look up a respondent in the database by name, id, or phone number. This may e.g. be used to change appointments, or inspect the data
- Do tracking of a respondent. Searching in population register or telephone register to find updated information about the respondent
- Look at a surveys questionnaire. To advise an interviewer or to be able to persuade an interviewer what they need to know about the survey
- Read survey's manual. Same as above
- Access to available web questionnaires

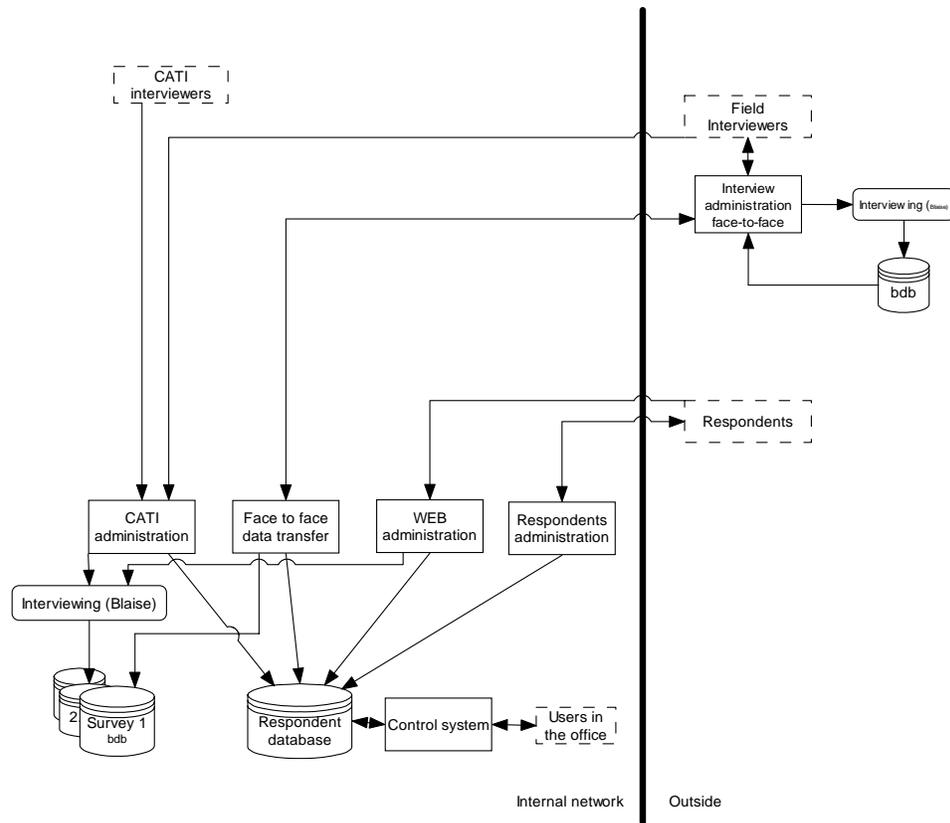
We have also some ideas for a respondent's page. The url of such a web page should be given in the letter to respondents. The page shall offer information about Statistics

Norway, about the survey and about privacy protection. It shall be possible for the respondent to give additional information e.g. telephone number on which the interviewer can reach him, and perhaps also time of day or week he is easiest to come in contact with. We want this information to update the respondent information in the database. Perhaps the web page after the data collection period can give some of the main results from the survey.

When the interviewers are out on face to face interviews they have to fill in a travel log in order to get money back for their travelling expenses. It's mostly driving expenses, so we have started to investigate a solution with GPS and logging, instead of manually filling in a form for travelling logs.

3.3 Sketch

As we see it now an overview over the data collection system will be something like this



3.4 More ideas

Hopefully we will have some more concrete ideas that we can present in September. The intention to present it in the conference is to get some feed back on our outlines, and perhaps get some additional ideas.

Contacts

Encrypting Blaise Data on Network Servers

John Mamer
Mathematica Policy Research, Inc., USA
JMamer@mathematica-mpr.com

Generic Data Storage with Blaise 4.8 Datalink

Arno Rouschen
Statistics Netherlands, Netherlands
ARON@CBS.nl

Methods for Simultaneous Meta & Data Manipulation in Blaise

Marien Lina
Statistics Netherlands, Netherlands
CLNA@CBS.nl

Blaise and Xml: Experiences Of An Outsourcing Project

Leif Bochis Madsen
Statistics Denmark, Denmark
lbm@dst.dk

Michigan Questionnaire Documentation System (MQDS): A User's Perspective

Heidi Guyer and Gina-Qian Cheung
The University of Michigan, USA
HGuyer@umich.edu

The use of mobile devices to carry out Blaise surveys at the ONS

Tim Burrell
Office for National Statistics, UK
tim.burrell@ons.gov.uk

Using the Blaise Alien Router For Audio-Recording of In-Person Interviews

M. Rita Thissen, Sridevi Sattaluri and Lilia Filippenko
RTI International, USA
rthissen@rti.org

Dynamic List Building Using Blaise

Gilbert Rodriguez and Barbara S. Bibb
RTI International, USA
bibb@rti.org

CTT: CAI Testing Tool

Mary Dascola, Genise Pattulo and Jeff Smith
The University of Michigan, USA
jefsmith@umich.edu

Blaise Testing

Rebecca Lui, Daniel Collison, Margaret Tang
Statistics Canada, Canada
Rebecca.Lui@statcan.ca

Case Scenario Library for Testing Large Blaise Applications

Rhonda Ash and Jason Ostergren
The University of Michigan, USA
rash@umich.edu

CATI Management: Behind the Scenes

Dave Dybicki
The University of Michigan, USA
ddybicki@umich.edu

Computer-Assisted Dialing: What will it do for you?

Dan Bernard
Marketing Systems Group, USA
DBernard@M-S-G.com

Using audit trails to monitor respondent behaviour in an Audio-CASI questionnaire

Olivier Bart
I.N.S.E.E., France
olivier.bart@insee.fr

Experiences Using an Event History Calendar in the Panel Study of Income Dynamics

April Beaulé, Eva Leissou and Youhong Liu
The University of Michigan, USA
abeaule@umich.edu

Retrospective Data Collection

Kate Cox, Elizabeth Hacker, Carli Lessof, Hayley Cheshire, Colin Miceli
National Centre for Social Research, UK
h.cheshire@natcen.ac.uk

Navigating BCP with .NET

Peter Sparks
The University of Michigan, USA
zebulon@umich.edu

Lifecycle Processes to Insure Quality of Blaise Interview Data

Linda Gowen, Pat Clark, and Jane Shepard
Westat, USA
SHEPHEJ1@WESTAT.com

The ScriptWriter Tool: An Application for Interviewer Training Script Development

Youhong Liu and Karl Dinkelmann
The University of Michigan, USA
yliu@umich.edu

Challenges in Converting the National Crime Victimization Survey to Blaise

Charlie Carter, Roberto Picha, Dan Moshinsky, Alexis Arrington, and Michael Mangiapane
US Census Bureau, USA
charlie.e.carter@census.gov

The ONS Integrated Household Survey: The next installment

Alessio Fiacco
Office for National Statistics, UK
Alessio.Fiacco@ons.gsi.gov.uk

Questionnaire Specification Database for Blaise Surveys

Lilia Filippenko, Joe Nofziger and Valentina Grouverman
RTI International, USA
lfilippenko@rti.org

Authoring Blaise questionnaires - A task for the survey specialist or an IT programmer?

Rebecca Gatward
Office for National Statistics, UK
Rebecca.Gatward@ons.gsi.gov.uk

Blaise Source Code Editing System

Sheila Deskins and Danilo Gutierrez
The University of Michigan, USA
rash@umich.edu

BLAISE in Macedonia

Liljana Taseva and Mira Deleva
State Statistical Office of Macedonia, Macedonia
ana.abjanic@stat.gov.mk

Designing and Developing Person-based and Topic-based Data Collection Instruments

Emily A. Johnson and Thomas C. Melaney
US Census Bureau, USA
emily.a.johnson@census.gov

Coping with people who just won't stay put: The Use of Blaise in Longitudinal Panel Surveys

Colin Setchfield
Office for National Statistics, UK
Colin.Setchfield@ons.gsi.gov.uk

Exploration of Blaise Instrument Generation from Metadata

Fred Wensing and Juanita Pettit
Australian Bureau of Statistics, Australia
juanita.pettit@abs.gov.au

Using Blaise 4.7 for data entry, checking, error reporting and transforming for further processing

Orhideja Krstanova
State Statistical Office of Macedonia, Macedonia
orhideja.krstanova@stat.gov.mk

Survey Specifications Management at Statistics Canada

Pamela Ford
Statistics Canada, Canada
Rebecca.Lui@statcan.ca

Analyses of Web Survey Data

Vesa Kuusela
Statistics Finland, Finland
Vesa.Kuusela@stat.fi

Web Application Stress Testing with Blaise IS

Jim O'Reilly
Westat, USA
JimOReilly@westat.com

Conversion of Blaise Databases to Relational Databases

Yogeeta Purohit, Latha Srinivasamohan, William Dyer
US Census Bureau, USA
yogeeta.r.purohit@census.gov

Deploying a C# .NET Object onto a Laptop to be used with a Blaise 4.7**Data Collection Instrument**

Mecene Desormice, Daniel Moshinsky, and Thomas C. Melaney
US Census Bureau, USA
thomas.charles.melaney@census.gov

Designing an e-Form to Collect Survey Data

David Kinnear
Office for National Statistics, UK
David.Kinnear@ons.gsi.gov.uk

Miscellaneous Examples of Programming in Blaise and Manipula

Rob Groeneveld
Statistics Netherlands, Netherlands
RGND@CBS.nl

New CAI operation at Statistics Norway

Hilde Degerdal and Jan Haslund
Statistics Norway, Norway
Hilde.Degerdal@ssb.no

