

The Future of Surveys for Official Statistics

Jelke Bethlehem, Statistics Netherlands, Methodology Department

1. The ever changing landscape of survey research

1.1 The role of Blaise in the world of survey research

The survey research landscape has undergone radical changes over the last decades. First, there was the change from traditional paper and pencil interviewing (PAPI) to computer-assisted interviewing (CAI). It started in the 1970's with Computer Assisted Telephone Interviewing (CATI). At that time the first CATI systems were developed by commercial market research agencies in the United States. CATI systems could not only handle interviewing by telephone from a centralised facility, but also took care of call scheduling and case management. CATI was followed by Computer Assisted Personal Interviewing (CAPI). CAPI emerged in the 1980's when lightweight laptop computers made face-to-face interviewing with a computer feasible. Self-administered forms of CAI also emerged during the 1980's. It was sometimes called in Computer Assisted Self Interviewing (CASI). The electronic questionnaire ran on a computer in the respondent's home. Respondents completed the questionnaire on their computer at home, after which the data were transmitted to the statistical agency. And now, particularly in commercial market research, face-to-face, mail and telephone surveys are increasingly replaced by web surveys. The popularity of Computer Assisted Web Interviewing (CAWI) is not surprising. Now that many people have Internet, a web survey is a simple means to get access to a large group of people.

The development of the Blaise System has followed the trends in survey research. Development of the system started in 1980. The first version came out in 1986. Blaise 1.0 was a CADI system. The basic idea was to create systems that could improve the handling of paper questionnaire forms by integrating data entry and data editing tasks. It ran under the operating system MS-DOS.

After version 1 of Blaise had been in use for a while, it was realised that the system could be made much more powerful by implementing computer assisted interviewing (CAI). Version 2.0 of Blaise was completed in 1988. It implemented just one form of computer assisted interviewing (CAPI). In 1990, another mode of computer assisted interviewing was included: Computer Assisted Telephone Interviewing (CATI). This was version 2.3 of Blaise.

When the operating system MS-DOS was gradually replaced by Windows, this had consequences for Blaise. Version 4 of Blaise marked the change from MS-DOS to Windows. Blaise 4 was the first Windows version of Blaise. It was released in 1998. The functionality of Blaise 4 was basically the same as that of the previous MS-DOS version. Of course, the graphical user interface offered much more possibilities for screen layout.

In the first 10 years of its existence, the basic concept of Blaise, the definition of the questionnaire in the Blaise language, turned out to be sufficiently powerful to cope with all developments in information technology and survey methodology.

Then came the rise of the Internet. As more and more people were connected to the Internet, computer assisted web interviewing (CAWI) gained in popularity amongst researchers as means of data collection. CAWI was implemented in Blaise 4.6. It was released in 2003. And again, the basic concept could cope with this new data collection technique without dramatic changes.

What will be the future developments in survey research? And what is needed in Blaise to keep up with these developments. This paper attempts to answer this question by pointing at some of these developments. After describing some history and future challenges of survey methodology, the rest

of the paper concentrates on two issues. The first one is mixed-mode data collection and the role web surveys can play in this. The second one is the collect data of acceptable quality under the constraint of a reduced survey budget. This may requires a new approach (such as responsive survey design) and new tools (such as the R-indicator).

1.2 Some history of survey research

The idea of conducting surveys for compiling statistical overviews of the state of affairs in a country is already very old, see Kendall (1960). As far back as Babylonian times censuses of agriculture were taken. Ancient China counted its people to determine the revenues and the military strength of its provinces. There are also accounts of statistical overviews compiled by Egyptian rulers long before Christ. The Roman Empire regularly took a census of people and of property. The data were used to establish the political status of citizens and to assess their military and tax obligations to the state. All these surveys were complete enumerations of the population. The idea of sampling had not yet emerged.

Censuses were rare in the Middle Ages. The most famous one was the census of England taken by the order of William the Conqueror, King of England. The compilation of this Domesday Book started in the year 1086. The book records a wealth of information about each manor and each village in the country.

Figure 1.2.1. RAPI: Rope Assisted Personal Interviewing



Another interesting example can be found in the Inca Empire that existed between 1000 and 1500 in South America. Each Inca tribe had its own statistician, called the Quipucamayoc. This man kept records of, for example, the number of people, the number of houses, the number of llamas, the number of marriages and the number of young men that could be recruited for the army. All these facts were recorded on a quipu, a system of knots in coloured ropes, see figure 1.2.1. A decimal system was used for this.

The idea of using sampling instead of a complete enumeration came up around the year 1895. In that year, Anders Kiaer (1895, 1997), the founder and first director of Statistics Norway, published his Representative Method. He proposed questioning only a (large) sample of persons were questioned who together formed a 'miniature' of the population. Anders Kiaer stressed the importance of representativity. His argument was that, if a sample was representative with respect to variables for which the population distribution was known, it would also be representative with respect to the other survey variables.

A basic problem of the Representative Method was that there was no way of establishing the accuracy of estimates. The method lacked a formal theory of inference. It was Bowley (1906, 1926), who made the first steps in this direction. He showed that for large samples, selected at

random from the population with equal probabilities, estimators had an approximately normal distribution.

From this moment on, there were two methods of sample selection. The first one was Kiaer's Representative Method, based on purposive selection, in which representativity played a crucial role, and for which no measure of the accuracy of the estimates could be obtained. The second was Bowley's approach, based on simple random sampling, and for which an indication of the accuracy of estimates could be computed. Both methods existed side by side for a number of years. This situation lasted until 1934, in which year the Polish scientist Jerzy Neyman published his now famous paper, see Neyman (1934). Neyman developed a new theory based on the concept of the confidence interval. By using random selection instead of purposive selection, there was no need any more to make prior assumptions about the population. Neyman also showed that the Representative Method based on purposive sampling failed to provide satisfactory estimates of population characteristics. As a result, the method of purposive sampling fell into disrepute in official statistics.

The principles of probability sampling formed to basis for modern survey taking. They are vital for making valid inference about the population being investigated.

These principles have been successfully applied in official and academic statistics since the 1940's, and to a much lesser extent also in more commercial market research. Where samples are not based on probability sampling, it is not possible to compute unbiased estimates, and it is also not possible to quantify margins of error.

What has changed over the years, are the instruments for actual data collection. Until the 1970's, all surveys used paper forms (PAPI) in either face-to-face, telephone or mail surveys. Then it became possible used computers for data collection. The paper form was replaced by a desktop or laptop computer. It simplified the work of the interviewers, produced higher quality data, and substantially reduced to time needed to carry out a survey. After the 1990's more and more people were connected to the Internet, and web surveys became possible. Conducting a survey was even more simpler, faster and cheaper.

1.2 Future trends

National statistical institutes in many countries are faced with three conflicting developments: (1) they face substantial budget constraints, (2) they are confronted with a demands for more and more detailed information, and (3) the have to reduce the response burden of surveys as much as possible. And of course, the quality of the published statistical information should remain at an acceptable level. Not surprisingly, many statistical are rethinking their data collection operations.

This paper explores some possible future directions for survey research to go. The first one is an increased use of web surveys. This is not without risks. Opportunities and threats are discussed in the first part of section 2. Another possible direction is to used web surveys as one of the modes in mixed-mode surveys. Several statistical organizations are already using mixed-mode survey or are planning to introduce mixed-mode surveys. Some of the challenges and also the implications for Blaise are discussed in in the second part of section 2.

Reducing survey costs and response burden would mean collecting as little data as possible, while maintaining an acceptable level of survey quality. This calls for adequate indicators of survey quality. It is argued in section 3 that the response rate is insufficient for this. A different indicator is proposed: the R-indicator. This indicator can also be used during data collection to monitor the fieldwork and make changes in the survey when necessary. This is called responsive survey design.

2. The conquest of the web

2.1 Single-mode web surveys

Web surveys have become increasingly popular over the last couple of years. This is not surprising. A web survey is a simple means to get access to a large group of people. Questionnaires can be distributed at very low costs. No interviewers are needed, and there are no mailing and printing costs. Surveys can be launched very quickly. Little time is lost between the moment the questionnaire is ready and the start of the fieldwork. And web surveys offer new, attractive possibilities, such as the use of multimedia (sound, pictures, animation and movies).

At first sight, online surveys seem to have much in common with other types of surveys. It is just another mode of data collection. Questions are not asked face-to-face or by telephone, but over the Internet. There are, however, major methodological issues. One issue is under-coverage. Since data are collected using the Internet, people without Internet access will never be able to participate in a web survey. This means research results can only apply to the Internet population and not to the complete population. Another issue is that sample selection is often based on self-selection of respondents instead of on probability sampling. Researchers have no control over the selection mechanism, resulting in unknown selection probabilities. Therefore, no unbiased estimates can be computed, nor can the accuracy of estimates be established. These problems are discussed below in some more detail.

Web surveys suffer from under-coverage because the target population is usually much wider than just the Internet population. According to data from Eurostat, the statistical office of the European Union, 54% of the households in the EU had access to Internet in 2007. There were large variations between countries. The countries with the highest percentages of Internet access were The Netherlands (83%), Sweden (79%) and Denmark (78%). Internet access was lowest in Bulgaria (19%), Romania (22%) and Greece (25%).

Even more problematic is that Internet access is unevenly distributed over the population. A typical pattern found in many countries is that the elderly, the low-educated and ethnic minorities are severely under-represented among those having access to Internet. Bethlehem (2007) shows that the bias of the response mean as an estimator of the population mean of a variable Y is equal to

$$B(\bar{y}) = E(\bar{y}) - \bar{Y} = \bar{Y}_I - \bar{Y} = \frac{N_{NI}}{N} (\bar{Y}_I - \bar{Y}_{NI}), \quad (2.1.1)$$

where the subscript I denotes the Internet population and NI the non-Internet population. The magnitude of this bias is determined by two factors. The first factor is the relative size N_{NI} / N of the sub-population without Internet. Therefore the bias decreases as Internet coverage increases. The second factor is the contrast $\bar{Y}_I - \bar{Y}_{NI}$ between the means of the Internet-population and the non-Internet-population. The more the mean of the target variable differs for these two sub-populations, the larger the bias will be. Since Internet coverage is steadily increasing, the factor N_{NI} / N is decreasing. This has a bias reducing effect. It is not clear, however, whether the contrast between the those with and without Internet also decreases. To the contrary, it is not unlikely that the (small) group of people without Internet will be more and more different from the rest of the population. As a result, substantial bias may still remain.

Application of self-selection will also cause estimates to be biased. Self-selection means that the survey is simply put on the web. Participation requires in the first place that respondents are aware of the existence of a survey. They have to accidentally visit the website, or they have to follow up a banner, e-mail message, or a call in another commercial. In the second place, they have to make the decision to fill in the questionnaire on the Internet. All this means that each element k in the

population has unknown probability ρ_k of participating in the survey. Bethlehem (2008) shows that the expected value of the sample mean is equal to

$$E(\bar{y}) \approx \bar{Y}^* = \frac{1}{N} \sum_{k=1}^N \frac{\rho_k}{\bar{\rho}} Y_k \quad (2.1.2)$$

where $\bar{\rho}$ is the mean of all response propensities. The bias of this estimator is equal to

$$B(\bar{y}) = E(\bar{y}) - \bar{Y} \approx \bar{Y}^* - \bar{Y} = \frac{R_{\rho Y} S_{\rho} S_Y}{\bar{\rho}}, \quad (2.1.3)$$

in which $R_{\rho Y}$ is the correlation coefficient of the target variable and the response probabilities, S_{ρ} is the standard deviation of the response probabilities, and S_Y is the standard deviation of the target variable. It can be shown that in the worst case (S_{ρ} assumes its maximum value and the correlation $R_{\rho Y}$ is equal to either +1 or -1) the absolute value of the bias is equal to

$$|B_{\max}(\bar{y})| = S_Y \sqrt{\frac{1}{\bar{\rho}} - 1}. \quad (2.1.4)$$

Bethlehem (1988) shows the formula (2.1.3) also applies in the situation in which a probability sample has been drawn, and subsequently nonresponse occurs during the fieldwork. Consequently, expression (2.1.4) provides a means to compare potential biases in various survey designs. For example, regular surveys of Statistics Netherlands are all based on probability sampling. Their response rates vary around 70%. This means the absolute maximum bias is equal to $0.65 \times S_Y$. One of the largest self-selection web surveys in The Netherlands was *21minuten.nl*. Within a period of six weeks in 2006 about 170,000 people completed the web questionnaire. The target population of this survey was not defined, as everyone could participate. If it is assumed the target population consists of all Dutch from the age of 18, the average response propensity is equal to $170,000 / 12,800,000 = 0.0133$. Hence, the absolute maximum bias is equal to $8.61 \times S_Y$. It can be concluded that the bias of the large web survey can be a factor 13 larger than the bias of the smaller probability survey.

The effects of self-selection can also be illustrated using an example related to the general elections in The Netherlands in 2006. Various survey organizations used opinion polls to predict the outcome of these elections. The results of these polls are summarized in table 2.1.1. Differences of two seats or more are printed in boldface. *Politieke Barometer*, *Peil.nl* and *De Stemming* were opinion polls carried out by market research agencies. They are all based on samples from web panels. To reduce a possible bias, adjustment weighting was carried out. The polls were conducted one day before the election. The Mean Absolute Difference indicates how big the differences (on average) are between the poll and the election results. Particularly, differences are large for the more volatile parties like PvdA, SP and the PVV. Differences of three seats or more are printed in boldface. For example, one poll predicted 32 seats in parliament for the SP (socialist party) whereas this party got only 25 seats. This difference is larger than could be explained by the sampling error.

Table 2.1.1. Parliamentary elections in The Netherlands (2006), predictions and results

	Election result	Politieke Barometer	Peil.nl	De Stemming	DPES 2006
Sample size		1,000	2,500	2,000	2,600
Seats in parliament:					
CDA (christian democrats)	41	41	42	41	41
PvdA (social democrats)	33	37	38	31	32

VVD (liberals)	22	23	22	21	22
SP (socialist)	25	23	23	32	26
GL (green party)	7	7	8	5	7
D66 (liberal democrats)	3	3	2	1	3
ChristenUnie (Christian)	6	6	6	8	6
SGP (Christian)	2	2	2	1	2
PvdD (animal party)	2	2	1	2	2
PVV (populist)	9	4	5	6	8
Other parties	0	2	1	2	1
Mean absolute difference		1.27	1.45	2.00	0.36

DPES is the Dutch Parliamentary Election Study. The fieldwork was carried out by Statistics Netherlands in a few weeks just before the elections. The probability sampling principle has been followed here. A true (two-stage) probability sample was drawn from the population register. Respondents were interviewed face-to-face (using CAPI). The predictions of this survey are much better than those based on the online opinion polls. The predictions and election results only differ for four parties, and differences are at most one seat.

Probability sampling has the additional advantage that it provides protection against certain groups in the population attempting to manipulate the outcomes of the survey. This may typically play a role in opinion polls. Self-selection does not have this safeguard. An example of this effect could be observed in the election of the 2005 Book of the Year Award (Dutch: NS Publieksprijs), a high-profile literary prize. The winning book was determined by means of a poll on a website. People could vote for one of the nominated books or mention another book of their own choice. More than 90,000 people participated in the survey. The winner turned out to be the new Bible translation launched by the Netherlands and Flanders Bible Societies. This book was not nominated, but nevertheless an overwhelming majority (72%) voted for it. This was due to a campaign launched by (among others) Bible societies, a Christian broadcaster and Christian newspaper. Although this was all completely within the rules of the contest, the group of voters could clearly not be considered to be representative of the Dutch population.

Can self-selection web surveys be used for data collection in official statistics? The discussion in this section leads to the conclusion that severe methodological problems make it very hard, if not impossible, to draw valid conclusions about the population to be surveyed. Self-selection can cause estimates of population characteristics to be biased. This seems to be similar to the effect of nonresponse in traditional probability sampling based surveys. However, it was shown that the bias in self-selection surveys can be substantially larger.

Self-selection is a serious problem, but it can be solved by applying probability sampling. A random sample (e.g. of addresses) can be drawn from a sampling frame. A letter can be sent to each selected address with request to complete a questionnaire on the Internet. Unique identification codes guarantee that the proper persons answer the questions. In fact, the only difference with a mail questionnaire is that the paper questionnaire form is replaced by an electronic one on the Internet.

The problem of under-coverage in web surveys has to be addressed too. It is interesting to note that only between 60% and 70% of the households in The Netherlands still have a listed landline telephone. So one out of three households is missing if a sample is selected from a telephone directory. This shows that also more traditional phone surveys suffer from under-coverage. It is expected that under-coverage for web surveys will decrease over time. From the point of view of coverage, a web survey may be better than a telephone survey, at least in The Netherlands.

If under-coverage in web survey really is a problem, a possible solution could be to simply provide Internet access to those without Internet. An example of this approach is the LISS panel, see

Scherpenzeel (2008). This online panel has been constructed by selecting a random sample of households from the population register of The Netherlands. Selected households were recruited for this panel by means of CAPI or CATI. So sample selection was based on true probability sampling. Moreover, co-operative households without Internet access were provided with equipment giving them access to Internet. Analysis by Scherpenzeel & Bethlehem (2010) shows that the results of this panel are closer to those of surveys based on probability sampling than to those based on self-selection web surveys.

Can a web survey be an alternative for a CAPI or CATI survey? With respect to data collection, there are substantial differences between CAPI and CATI on the one hand and web surveys on the other. Interviewers carry out the fieldwork for CAPI and CATI surveys. They are important in convincing people to participate in the survey, and they also can assist in completing the questionnaire. There are no interviewers in a web survey. It is a self-administered survey. Therefore quality of collected data may be lower due to higher nonresponse rates and more errors in the answers to the questions. However, response to sensitive questions may be higher and better without interviewers.

2.2 Mixed-mode surveys

Budget cuts on the one hand and demands for more and more detailed information on the other, while maintaining an acceptable level of data quality, have stimulated statistical agencies to explore different approaches to data collection. One such approach is the mixed-mode survey. Different data collection modes are used in such a survey.

De Leeuw (2005) describes two mixed-mode approaches. The first approach is to use different modes concurrently. The sample is divided into groups and each group is approached by a different mode. The other approach is to use different modes sequentially. All sample persons are approached by one mode. The non-respondents are then followed up by a different mode than the one used in the first approach. This process can be repeated for a number of modes.

If cost reduction is the main issue, one could think of a mixed-mode survey that starts with a questionnaire on the web. Non-respondents are followed up by CATI. Non-respondents remaining after CATI could be followed up by CAPI. So the survey starts with the cheapest mode and ends with the most expensive one.

If quality and response rate are of vital importance, one could think of a mixed-mode design that starts with CAPI. The non-response is followed-up by CATI. Remaining non-respondents are asked to complete the questionnaire on the web.

Mixed-mode survey suffer from mode effects. Mode effects occur if the same question produces a different answer when asked in a different mode. The presence or absence of interviewers may be a source of mode effects. The presence of interviewers leads to more socially desirable answers, particularly for questions about potentially embarrassing behaviour. The presence of interviewers also causes acquiescence. This is the tendency to agree with statements by interviewers. It is easier to agree than to disagree.

The interviewers are in control of presenting the questions to the respondents in CAPI and CATI surveys. They can see to it that the respondents hear and understand every word of it. When necessary, additional explanation can be provided. This is different for self-completing surveys. There is on guarantee that questions are carefully read and clearly understood.

There are also mode effects with respect to answering closed questions. Research seems to suggest that respondents in mail or web surveys more often choose the first answer option (primacy effect), while there is a preference for the last answer option in CAPI and CATI surveys (recency effect).

A final mode effect to be mentioned here is caused by the treatment of “don’t know”. This option is often not offered explicitly in CAPI or CATI surveys, but the interviewers have a facility to record this answer if the respondents insist they really do not know the answer. For self-administered surveys, this option is either explicitly offered or it is not possible at all to give this answer. If ‘don’t know’ is clearly one of the answer options, more respondents will select this option (the easy way out).

It will be clear that a mixed-mode survey may suffer from mode-effects. There are two approaches to reduce these effects. One is to develop separate questionnaires for different modes. A specific question may be defined differently in different modes as long as it measures the same thing. The different versions of the question should be cognitively equivalent. This is not very easy to realise, as it may take substantial research and experimentation. Moreover, it may turn out to be cumbersome and error prone to maintain three different Blaise questionnaires for one survey.

Dillman (2008) proposed his so-called unimode approach. This is a set of guidelines to define questions in such a way that the mode effects are minimized. Here are some examples of guidelines:

- Keep all answer options the same across modes.
- Include all answer options in the text of the question.
- Reduce the number of answer options as much as possible.
- Reverse the order of the answer options in half of the questionnaires.
- Develop equivalent instructions for skip patterns

Instead of reversing the order of the answer options in half of the questionnaire one could also think of randomizing the order of the answer options. It may be difficult, or even impossible, to implement equivalent skip instructions for paper questionnaires.

One may wonder whether it is possible to develop a questionnaire which completely satisfies all unimode guidelines. Particularly of attitudinal questions, it may turn out to be necessary to define mode-dependent versions. This may lead to one Blaise questionnaire, but with rule instructions like

```
Question_A
IF Mode = CAPI THEN
    Question_B1
ELSEIF Mode = CATI THEN
    Question_B2
ELSEIF Mode = CAWI THEN
    Question_B3
ENDIF
Question_C
```

A final aspect of mixed-mode surveys to be mentioned here, is case management. This is of vital importance, particularly in case of a sequential mixed-mode approach. A case management system should see to it that cases are assigned to the proper mode at the proper moment. Cases may not disappear from the system. Also, duplicate cases must be avoided. This calls for an overall case management system.

3. The quest for representativity

3.1 The response rate as a quality indicator?

Most surveys suffer from non-response. This is the phenomenon that sample elements do not provide the required information. Non-response may seriously affect the quality of the outcomes of a survey. Estimates of population characteristics will be biased if, due to non-response, some groups in the population are over- or underrepresented, and these groups behave differently with respect to the survey variables.

Survey agencies often use the survey response rate as an indicator of survey quality. However, a low response rate does not necessarily imply that the accuracy of survey estimates is poor. If non-response is ignorable, i.e. there is no direct correlation between response behaviour and the survey variables, estimates will still be unbiased. Indeed, the literature on survey methodology contains ample examples showing that response rates by themselves are poor indicators of non-response bias. As an indicator of survey quality it can be misleading.

This is illustrated by an example from the 1998 Dutch POLS survey (short for Permanent Onderzoek Leefsituatie or Integrated Survey on Household Living Conditions in English). Table 3.1.1 contains estimates of two population quantities: the percentage of people receiving some form of social allowance and the percentage of people having at least one parent that was born outside the Netherlands. The people are, by definition, non-native. Both variables are taken from a register and are artificially treated as survey questions. Therefore sample percentages are also available. These sample percentages are given in table 3.1.1. After one month of fieldwork the response rate was 47.2%, while after the full two month period the rate had increased to 59.7%. The mode of data collection in the first month was CAPI (Computer Assisted Personal Interviewing). Non-respondents were approached in the second month with CATI (Computer Assisted Telephone Interviewing) if they had a listed, land-line phone. Otherwise, CAPI was used again. The second month of fieldwork increased the response by 12.5% This did, however, not result in better estimates. The bias of the estimators increased after the second month.

Table 3.1.1. Response means in POLS after the first and second month of data collection

Variable	After 1 month	After 2 months	Sample
Social allowance	10.5 %	10.4 %	12.1 %
Non-native	12.9 %	12.5 %	15.0 %
Response rate	47.2 %	59.7 %	100.0 %

There is a need for additional survey quality indicators that provide more insight in the possible risk of biased estimators. This section describes such an indicator. It is called the *R-indicator*. The R stands for 'representativity'. R-indicators measure how representative the survey response is, or to say it differently, how the composition of the response differs from that of the sample.

R-indicators can be used in many different ways. One way is to inspect the survey data after completion of the fieldwork. But they can also play an important role during data collection. By monitoring the fieldwork, data collection efforts can be targeted at obtaining a response the composition of which does not deviate too much from that of the complete sample (or the population).

3.2 What is representativity?

The concept of representativity is often used in survey research, but usually it is not clear what it means. Kruskal and Mosteller (1979a, 1979b and 1979c) present an extensive overview of what representative is supposed to mean in non-scientific literature, scientific literature excluding sta-

tistics and in the statistical literature. They found the following meanings for ‘representative sampling’: (1) general acclaim for data, (2) absence of selective forces, (3) miniature of the population, (4) typical or ideal case(s), (5) coverage of the population, (6) a vague term, to be made precise, (7) representative sampling as a specific sampling method, (8) as permitting good estimation, or (9) good enough for a particular purpose. They recommended not using the word *representative*, but instead to specify what one means.

To be able to define an indicator for representativity, the concept of representativity is defined here as the absence of selective forces. Every element k in the population is assumed to have a certain, unknown, probability ρ_k of responding when selected in the sample. It is clear that there are no selective forces if all response probabilities are equal. Unfortunately, response probabilities are unknown in practice. Therefore they have to be estimated using the available data. To this end, the concept of the response propensity is introduced. The *response propensity* of element k is defined by

$$\rho_k(X) = P(R_k = 1 | X_k), \quad (3.1.1)$$

where $X_k = (X_{k1}, X_{k2}, \dots, X_{kp})'$ is a vector of values of auxiliary variables. The response indicator R_k assumes the value 1 if element k is selected in the sample and responds; otherwise R_k assumes the value 0. So the response propensity is the probability of response given the values of some auxiliary variables. The response propensities are also unknown, but they can be estimated provided the values of the auxiliary variables are available for both the respondents and non-respondents. To be able to estimate the response propensities, a model must be chosen. The most frequently used one is the logistic regression model. It assumes the relationship between response propensity and auxiliary variables can be written as

$$\log it(\rho_k(X)) = \log \left(\frac{\rho_k(X)}{1 - \rho_k(X)} \right) = \sum_{j=1}^p X_{kj} \beta_j, \quad (3.1.2)$$

where $\beta = (\beta_1, \beta_2, \dots, \beta_p)'$ is a vector of regression coefficients. The logit transformation ensures that estimated response propensities are always in the interval $[0, 1]$.

3.3 The R-indicator

The R-indicator measures how far the composition of the response to a survey deviates from the original sample. If all response probabilities are equal, the response is representative, and there will be no systematic differences between the composition of the response and the sample. If the response probabilities are not equal, it is important to establish to what extent the composition of the response is affected. This is accomplished by defining a distance function that measures how far the individual response probabilities differ from the mean response probability.

Suppose, that the individual response probabilities $\rho_1, \rho_2, \dots, \rho_N$ of all elements in the population are known. Then the standard deviation

$$S(\rho) = \sqrt{\frac{1}{N-1} \sum_{k=1}^N (\rho_k - \bar{\rho})^2}, \quad (3.3.1)$$

of the response probabilities can be computed. This is a distance function. $S(\rho) = 0$ if all response probabilities are equal, and the value of $S(\rho)$ will be larger as there is more variation in the values of the response probabilities. One can prove that the maximum value of $S(\rho)$ is equal to 0.5. The R-indicator is now be defined as

$$R(\rho) = 1 - 2S(\rho) \quad (3.3.2)$$

This R-indicator assumes a value in the interval [0, 1]. A value of 1 implies strong representativity. The smaller its value is, the more the response composition deviates from that the sample composition.

The values of the individual response probabilities are unknown in practice. This is solved by estimating response propensities as defined in (3.1.1), for example with a logit model. Estimates $\hat{\rho}_1, \hat{\rho}_2, \dots, \hat{\rho}_n$ can only be computed for the sample elements. The R-indicator (3.3.2) can now be estimated by

$$\hat{R}(\rho) = 1 - 2 \sqrt{\frac{1}{N-1} \sum_{i=1}^n \frac{(\hat{\rho}_k - \hat{\rho})^2}{\pi_i}}, \quad (3.3.3)$$

where π_i is the first order inclusion probability of sample element i . Note that expression (3.3.3) involves two estimation steps. The first one is estimation of the response probabilities and the second one is estimation of the standard deviation.

The R-indicator was applied in a large scale follow-up study among the non-respondents in the Dutch Labour Force Survey (LFS) in 2005. Two samples of non-respondents were approached once more using either a call-back approach with the full LFS questionnaire or a basic-question approach with a very short questionnaire containing only a few basic questions. Some results are summarised in table 3.3.1. For more details see Schouten (2007) and Cobben and Schouten (2007). The R-indicator was estimated using logistic regression models including a large number of explanatory variables that measured demographic, geographic and socio-economic characteristics of the households.

Table 3.3.1. Comparing R-indicators in the LFS follow-up study

Response	Response rate	R-indicator
LFS	62.2 %	0.80
LFS + call-back approach	76.9 %	0.85
LFS + basic-question approach	75.6 %	0.78

The value of the R-indicator for the initial LFS response is equal to 0.80, which is lower than the ideal value of 1.00. So this response is not completely representative. Application of the call-back approach increases the response rate from 62.2% to 76.9%. The value of the R-indicator also increases, from 0.80 to 0.85. This indicates that the additional response improves the composition of the data set. Application of the basic-question approach results in a different conclusion. Although the response rate increases from 62.2% to 75.6%, the value of the R-indicator drops from 0.80 to 0.78. Apparently, the basic-question approach does not improve the composition of the data set. This approach gives ‘more of the same’ and, hence, sharpens the contrast between respondents and non-respondents.

3.4 Use of the R-indicator

The R-indicators can be used in different ways in the survey process. A number of possibilities are described here:

- Monitoring the survey process. Already during data collection it can become clear whether or not the composition of the collected data differs from that of the initial sample. The outcomes of this monitoring process may help to underpin a decision to initiate additional efforts to obtain data for specific groups in the target population.

- Controlling the survey process. Use of an R-indicator already during the data collection phase may reveal that the composition of the collected data may deviate more and more from representativity. This could lead to a decision to focus the remainder of the data collection process on groups that are under-represented. Groves & Heeringa (2006) call these mid-survey decisions to change the design “responsive survey design”. See also section 3.5.
- Selection of auxiliary variables for non-response correction. Estimation of response probabilities is based on models involving auxiliary variables. Variables that significantly contribute to predicting response probabilities are also important in non-response correction techniques like adjustment weighting.
- Analysis of surveys. The R-indicator can be used as a simple analysis tool providing insight in possible problems due to non-response. Like the response rate, it is a quality indicator. The R-indicator can also be very useful for comparing surveys over times or comparing survey data for different domains, regions or countries.

The R-indicator proposed in this paper is promising because it can be estimated using sample data and it allows for easy interpretation. Computation of its value is reasonably straightforward with standard software like SPSS, SAS or STATA. If the R-indicator is to be used for monitoring or controlling the survey process, the data collection system used must be able to compute the R-indicator ‘on the fly’.

Research with respect to the R-indicator is still in progress. It is the objective of the RISQ project to develop and to test the R-indicator. RISQ stands for Representativity Indicators for Survey Quality. Five partners participate in this European project: Statistics Netherlands, Statistics Norway, The Statistical office of Slovenia, the University of Southampton (UK) and the University of Leuven (Belgium). The RISQ project is financed by the 7th Framework Programme of the European Union. More information can be found on www.r-indicator.eu.

3.5 Responsive survey design

Statistical agencies in many countries experience decreasing response rates in surveys. There seems to be a growing reluctance to participate in surveys. Efforts to keep response at an acceptable level increases survey costs. Decreasing response rates also affect the quality of survey results. There is a higher risk of biased estimates of population characteristics. This trend calls for a new approach. Groves & Heeringa (2006) propose an approach called *responsive survey design*.

Traditional survey designs are fixed. Aspects like sampling design, mode of data collection, respondent recruitment protocols, number of call-backs are all decided in the design phase, and never changed. However, it may turn out during the data collection process that these decisions are not the best ones, and that changes are required in order to obtain reliable statistics. This is the idea behind response survey design.

According to Groves & Heeringa (2006), the fieldwork of the survey is assumed to consist of a number of phases. The survey design features can be different in each phase. For example, the mode of data collection in the first phase could be CAPI, whereas a different could be used in subsequent phases. Responsive survey design consists of the following four steps:

- 1) Identify survey design features that potentially affect costs and quality of the survey. Typical examples of such features are the mode of data collection, sample size, and number of call-back attempts;

- 2) Define a set of indicators to measure (during data collection) the survey design features identified in step 1. Typical indicators are the response rate, the R-indicator described in section 3.4, and interviewer costs.
- 3) Measure the cost and quality indicators in each phase. Decide at the end of a phase to change the design features of the next phase based on the values of the indicators.
- 4) At the end of the fieldwork, combine the data collected in the separate design phases to obtain single estimators for population characteristics.

So the difference with traditional survey design is that during the fieldwork decisions may be taken to change the fieldwork based on information obtained during the fieldwork. This approach resembles to some extent the ideas about quality control that were proposed by Deming (1986) in his famous book on improving quality and productivity in industry. Many of his famous 14 points for management also apply to the production of statistical information. One of these points states that one should cease dependence on mass inspection. Inspection of the final product to improve quality is too late, ineffective and costly. Quality must be built in at the design stage. These statements particularly apply to data collection. By trying to detect and correct problems in the fieldwork well after they have occurred, one fails to locate the source of these problems, and consequently, these problems can not be solved.

A responsive survey design can only be implemented if a sufficient amount of information about the data collection process becomes available during the data collection process. Fortunately, computer assisted interviewing systems like Blaise are capable to produce these so-called paradata. Tools have to be developed that implement the cost and quality indicators discussed above.

Of course, the survey data collection as a whole has to be capable of implementing responsive survey design. Again this requires a case management system that is sufficiently powerful to transfer cases between data collection modes.

More about the topic of responsive design can be found in Groves & Heeringa (2006), Mohl & Laflamme (2007) and Wagner & Raghunathan (2007).

4. Some conclusions

To prepare Blaise for the future in official statistics, attention should be paid to developments in survey methodology. These developments are partly triggered by increasing nonresponse rates and demands for reduction of costs and response burden. These developments are visible world-wide.

Even in the early days of Blaise, the system was already presented as an integrated system for survey processing. For example, Bethlehem (1990) stressed the importance of consistency between data collection modes that was enforced by Blaise. This makes it very suitable for implementing mixed-mode surveys with the Dillman's unimode approach. What is lacking, is a set of tools or system to implement an effective case management system. Of course, it is possible to develop a tailor-made system for each separate mixed-mode survey, but that requires substantial efforts and resources. A generalized case management system should recognize that different survey organizations may have different requirements. Therefore it may be a challenge to develop a standard system. Blaise already has a CATI call management system. Possibly their experiences with this system could help.

The literature on survey methodology research shows there is a demand for tools to monitor the data collection process. The R-indicator is such a tool. And more are needed to implement responsive survey design. Such tools should be able to access Blaise data and paradata files. It is to

be expected that some of these tools require complex computations, like maximum likelihood estimation. It may turn out that a tool like Manipula is not implement these indicators. Of course, it is always possible to develop tools as independent programs, like Bascula. To give survey researchers the possibility to experiment with new tools, it would be nice to have a direct link between Blaise and, for example, the R language. R is an open source language and environment for statistical computing and graphics. Many powerful statistical techniques have already been implement in R, see www.r-project.org.

5. References

- Bethlehem, J.G. (1988), Reduction of nonresponse bias through regression estimation. *Journal of Official Statistics* 4, pp 251-260.
- Bethlehem, J.G. (1990), The Blaise System for Integrated Survey Processing. CBS-report, Netherlands Central Bureau of Statistics, Voorburg, The Netherlands.
- Bethlehem, J.G. (2007), *Reducing the bias of web survey based estimates*. Discussion Paper 07001, Statistics Netherlands, Voorburg/Heerlen, The Netherlands.
- Bethlehem, J.G. (2008), *How accurate are self-selection web surveys?* Discussion Paper 08014, Statistics Netherlands, The Hague/Heerlen, The Netherlands.
- Bowley, A.L. (1906), Address to the Economic Science and Statistics Section of the British Association for the Advancement of Science. *Journal of the Royal Statistical Society* 69, pp. 548-557.
- Bowley, A.L. (1926): Measurement of the precision attained in sampling. *Bulletin of the International Statistical Institute*, XII, Book 1, pp. 6-62.
- CBS (1987), *Automation in Survey Processing*, CBS Select. Staatsuitgeverij, The Hague, The Netherlands..
- Cobben, F. & Schouten, B. (2007), *A follow-up with basic questions of nonrespondents to the Dutch Labour Force Survey*. Discussion paper 07011, Statistics Netherlands, Voorburg, The Netherlands.
- Couper, M.P., Baker, R.P., Bethlehem, J.G., Clark, C.Z.F., Martin, J., Nicholls II, W.L., O'Reilly, J.M. (eds.) (1998), *Computer Assisted Survey Information Collection*. John Wiley & Sons, New York, USA.
- De Leeuw, E.D.(2005), To mix or not to mix data collection modes in surveys. *Journal of Official Statistics*, Vol. 21, No. 2, pp. 233 – 255.
- Deming, W.E. (1986), *Out of the crisis*. Cambridge University Press, Camebridge.
- Dillman, D.A., Smyth, J.D. & Christian, L.M. (2008), *Internet, Mail, and Mixed-Mode Surveys, The Tailored Design Method*. John Wiley & Sons, Hoboken, USA.
- Groves, R.M. and Heeringa, S.G. (2006), Responsive design for household surveys: tools for actively controlling survey errors and costs. *Journal of the Royal Statistical Society: Series A*, 169, pp. 439 – 457.
- Kendall, M.G. (1960), Where shall the history of statistics begin? *Biometrika*, 47, pp. 447-449.

Kiaer, A. N. (1895), Observations et expériences concernant des dénombrements représentatives. *Bulletin of the International Statistical Institute*, IX, Book 2, pp. 176-183.

Kiaer, A. N. (1997 reprint): *Den repräsentative undersøkelsesmetode*. Christiania Videnskabselskabets Skrifter. II. Historiskfilosofiske klasse, Nr 4 (1897). English translation: The Representative Method of Statistical Surveys, Statistics Norway.

Kruskal, W. & Mosteller, F. (1979a), Representative sampling I: non-scientific literature. *International Statistical Review* 47, pp. 13-24.

Kruskal, W. & Mosteller, F. (1979b), Representative sampling II: scientific literature excluding statistics. *International Statistical Review* 47, pp. 111-123.

Kruskal, W. & Mosteller, F. (1979c), Representative sampling III: current statistical literature, *International Statistical Review* 47, pp. 245-265.

Mohl, C. & Laflamme, F. (2007), Research and responsive design options for survey data collection at Statistics Canada. Proceedings of the American Statistical Association, Section on Survey Research Methods, pp.2962-2968.

Neyman, J. (1934), On the two different aspects of the representative method: the method of stratified sampling and the method of purposive selection. *Journal of the Royal Statistical Society* 97, pp. 558-606.

Nicholls, W.L. & Groves, R.M. (1986), The status of computer assisted telephone interviewing. *Journal of Official Statistics* 2, pp. 93-134.

Scherpenzeel, A. (2008), An online panel as a platform for multi-disciplinary research. In: I. Stoop & M. Wittenberg (eds.), *Access Panels and Online Research, Panacea or Pitfall?* Aksant, Amsterdam, pp. 101-106.

Scherpenzeel, A. & Bethlehem, J. (2010), How representative are online-panels? Problems of coverage and selection and possible solutions. In: M. Das, P. Ester, L. Kaczmirek & P. Mohler (eds.), *Social Research and the Internet: Advances in applied Methods and New Research Strategies*. Routledge Academic, New York, USA. To be published.

Schouten, B. (2007), *A follow-up of nonresponse in the Dutch Labour Force Survey*. Discussion paper 07004, Statistics Netherlands, Voorburg, The Netherlands.

Wagner, J. & Raghunathan, T. (2007), Bayesian approaches to sequential selection of survey design protocols. Proceedings of the American Statistical Association, Section on Survey Research Methods, pp.3333-3340.

Converting to Blaise 4.8 for CATI and CAWI Surveys

Leonard Hart and Scott Reid, Mathematica Policy Research, Inc.

Blaise 4.8 introduced new features, such as a revised computer-assisted telephone interviewing (CATI) call scheduler, a revamped version of Blaise Internet Services (Blaise IS) and a client/server method of data manipulation and storage. This paper will review the experience of Mathematica Policy Research, Inc. (MPR) in implementing Blaise 4.8 for operational surveys.

In addition to the challenges converting from Blaise 4.7 to Blaise 4.8, the conversion enabled us to switch to a new network operating system, which made it possible for us to implement a new internet authentication system and upgrade to new internet and data storage/CATI servers. This conversion to Blaise 4.8 also required us to adjust some of our operational procedures for installing and maintaining instruments in production.

1. Background

The core survey data collection goals at MPR are to find tools that enable us to collect data as efficiently as possible, especially in a multimode (CATI, computer-assisted web interviewing [CAWI], computer-assisted data entry [CADE], computer-assisted personal interviewing [CAPI]) environment and meet the ever-changing data collection requirements of our clients. With advances in computer technologies and architecture, changing thoughts in the area of survey methodology, and the speed with which respondents learn and accept technology changes in their day-to-day lives, meeting these goals is a never-ending challenge.

Ideally, every company relying on data collection software has the goal of a unimode instrument capable of handling multimode surveys while collecting data in one centralized database in real time. Previous to Blaise 4.8, MPR made great strides in this direction, but we did not rely on Blaise IS to fulfill this goal. As Blaise 4.8 was being released for beta testing, we decided to take a hard look at Blaise IS. Previous versions of Blaise IS could not handle a significant number of concurrent users, nor did they meet MPR's criteria of utilizing a multimode, real-time shared database.

MPR hoped to move forward from Blaise 4.7 to 4.8 for several other reasons. First, we wanted to replace an older CAWI product called C2B (from the University of Tillburg, Netherlands) that was no longer supported, used older technology, was getting harder for us to maintain, and had programming and security limitations. Next, we felt MPR would benefit by utilizing some the new features in 4.8, such as Client/Server-based communications, improvements to the CATI suite of programs, and possibly utilizing a more robust version of DataLink, which would enable us to centralize our data into a common SQL database. Lastly, MPR needed to replace its existing internet survey authentication system (iChain from Novell) due to the high cost of maintaining that system. This change presented us with the opportunity to upgrade to the Microsoft/ASP.Net solution of Full Content Protection (FCP) for our security and authentication needs.

2. Preliminary Goals

Our first goal in testing Blaise 4.8 was to determine if it was capable of meeting MPR's current interviewing needs. We initially started by testing the out-of-the-box examples Statistics Netherlands (SN) provided to see if we could run them within our current environment. We built upon these tests by adapting some of our existing programs to work under 4.8.

If Blaise 4.8 passed our initial tests, we planned to look for upcoming projects to run under this version. We found three projects targeted for production that provided us with the chance to test a CATI-only survey as well as one with CATI and CAWI in the same instrument. Over the next few months, with these three projects in mind, we began to set up the infrastructure necessary to move forward and begin testing Blaise 4.8.

Using Blaise 4.8 for Census Coverage Measurement

Roberto V. Picha, U.S. Census Bureau, USA

1. Introduction

This paper discusses a few of the innovations made to the Census Coverage Measurement Person Interview (CCM PI) survey instrument developed by the U.S. Census Bureau using Blaise 4.8. The original instrument was developed in preparation for the 2006 Dress Rehearsal using Blaise 4.6. As the 2010 Decennial Census operations approached, the Technologies Management Office Authoring Staff was requested to program the 2009 CCM PI Dress Rehearsal instrument by starting with the 2006 instrument and adding enhancements to it. The updated instrument specs contained new sets of requirements that presented challenges for the Authoring Staff.

This paper covers how the following functionality was enhanced or added to the 2009 CCM PI Dress Rehearsal instrument:

- ❑ Collecting, Displaying and Selecting Addresses
- ❑ Output Processing
 - Roster Collections and Roster Review
 - Navigating to the Last Question Asked From a Previous Session

2. Background

The CCM PI is conducted by personal visit and telephone using a computer-assisted data collection instrument on a laptop computer. The purpose of the CCM PI is to obtain person and address information about current residents of the sample housing unit, residents who may have moved since Census Day, and residents who may have moved out of the sample housing unit between Census Day and the time the CCM PI interview is conducted. The instrument collects demographic information for each person in the sample housing unit including name, gender, age, date of birth, race, and Hispanic origin. Information is also collected to determine where each current resident was living on Census Day and where each resident who has moved out since Census Day currently lives. Lastly, the instrument also collects information to determine if there are any other addresses where residents may have been counted on Census Day and other information necessary to geocode their alternate addresses. This data will then be compared to the decennial data in order to measure the accuracy of the person count of the 2010 Census.

In addition to developing the CCM PI instrument, a CCM Re-Interview (RI) instrument was created for verifying how the PI was conducted and will actually run the full PI if needed.

The CCM PI survey sample size is extremely large and will be implemented by Field Representatives (FRs) that have been temporarily hired to conduct this interview. Therefore, the premise for some of the CCM PI requirements is that the FRs collecting data do not have the usual experience, understanding, and training of current CAPI FRs. To this end, some of the requirements requested and implemented are ones that are not normally allowed in our current survey instruments.

Examples of these types of requirements are:

- ❑ Allowing empty responses for questions that should be answered
- ❑ Helping the FR with navigation upon re-entry
- ❑ Forcing forward navigation and locking complete sections that would alter the data collection if changed.

3. Collecting, Displaying, and Selecting Addresses

One of the more challenging requirements for the 2006 CCM PI instrument was to collect, display all collected addresses, and allow the FR to select an address from a list of up to 30 addresses per case. Each unique address entered for the various household members was added to an external ASCII file. The contents of this file were presented to the FR in the form of an answer list so the FR could select a previously entered address. When completing the address collection portions of the instrument, the FR either selected one from this list or added a new address. This same list of addresses was constantly updated and kept unique throughout the various sections in the instrument.

In the CCM PI instrument, an address is composed of seven components: house number, street name, unit designation, city, state, zip code, and country. Each unique address can be shared across different types of residencies throughout various sections of the instrument. In general, the FR can select an address previously entered in that section or another section for the same case. If the FR selects “Add New Address” at the Select Address question then all components will be available for editing. Otherwise, if the FR selects a previously entered address from the answer list (which displays all unique addresses entered for the case) the information displayed is transferred to each address component and then left as “show” only.

3.1 How Address Collection was accomplished in 2006

The 2006 CCM instrument attached an alien router written in Delphi that was exclusively built to write the address components collected through the Data Entry Program to an external file. Each unique address was written to an ASCII file by this alien router. In order for the alien router to work properly, the 2006 CCM instrument implemented a concept called gates. The gates were extra questions that triggered some events inside the alien router. The entry and exit gates were placed around the address components. Some times there were two gates one after another; the purpose of these gates was to turn on and off the writing of the external file and avoid infinite writing of the same address.

The addition of these gates presented a challenge to the FR of extra, unnecessary, keystrokes needed when collecting data. This was unacceptable from the survey Methodologists perspective; however, the use of gates was considered unavoidable and left in place for 2006.

The graphic below demonstrates one type of gating utilized. The “Correct” questions locked the address information entered in the components. The locking was accomplished by setting flags in the section to disable the address information from editing.

In order to modify the entries made in any of the address information the FR had to change the value of the “Correct” variable and manually navigate to where the change was needed.

Is the address you selected or entered correct?

1. Yes
 2. No

Name	Select	Probe	House#	Street	Unit	City	State	Zip	US	Description	Landmarks	Correct	Continue
Joe Doe	<input type="checkbox"/>	<input type="checkbox"/>	111	main	21	New ctt	Maine	22344	1			<input type="checkbox"/>	<input type="checkbox"/>
Jane Doe	<input type="checkbox"/>	<input type="checkbox"/>										<input type="checkbox"/>	<input type="checkbox"/>
Jill Doe	<input type="checkbox"/>	<input type="checkbox"/>										<input type="checkbox"/>	<input type="checkbox"/>

The “Continue” question enabled the next row if the next resident had an address in this section; otherwise the FR was taken to the next section.

Name	Select	Probe	House#	Street	Unit	City	State	Zip	US	Description	Landmarks	Correct	Continue
Joe Doe	0		111	main	21	New ctt	Maine	22344	1			1	
Jane Doe													
Jill Doe													

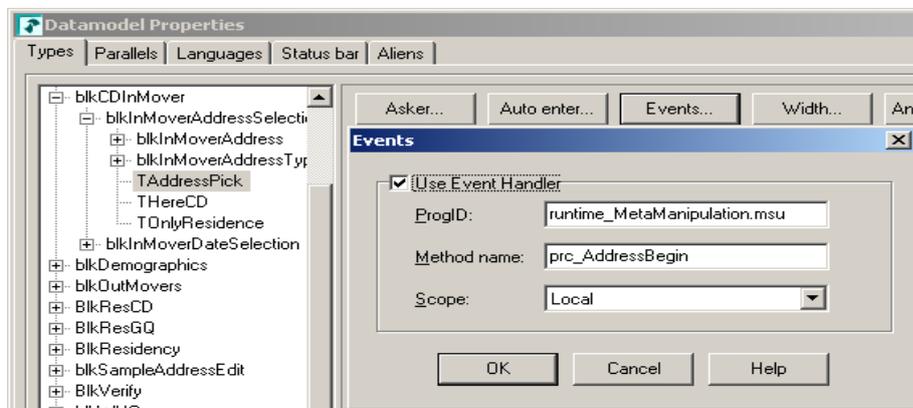
3.2 How Address Collection was implemented in 2009 using Blaise 4.8

For the new 2009 instrument, our sponsor wanted to make address collection more efficient and user-friendly by eliminating the extra questions added for gating the address components. This imposed a series of challenges to the authoring team. First, the original developers that worked with the Delphi application were no longer in the Division and secondly, our team no longer had a copy of this software.

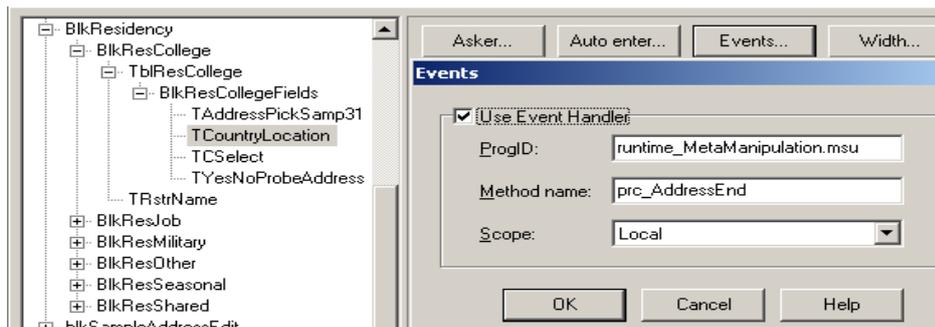
Since Blaise 4.8 offers a robust capability to allow the Data Entry Program to interact with Manipula, a decision was made to migrate the original code written in Delphi to Manipula. This approach proved to be successful and straightforward, obtaining the same results as the original approach plus the additional new requirements requested by our sponsors. They requested that the 2009 CCM PI instrument filter addresses based on the section in the instrument and compare addresses for possible duplicates among other addresses already entered in the various sections of the instrument.

The CCM PI is composed of about twenty-five address sections, each of which collects the same type of address components. This similarity made the implementation of the Manipula script possible. In order to implement the changes, the team removed the four references to the DLL Delphi code and the extra gate fields. We then applied the new call to a Manipula script as an alien procedure. To do this, three questions were mapped to make the call and replace the function of the gates from the original instrument. The first gate was replaced in the “Select Address” question with Datatype TAddressPick, the second in the “probe” question with Datatype TyesNoProbeAddress and the third in the last address component “Country” question with Datatype TCountryLocation.

The graphic below is an example of how to attach to a Manipula script via the data model properties and associate to a data type used in the Answer List for selecting an address.



The same concept was used for writing to an ASCII file in the Country question as shown below.



The following screen demonstrates the new layout and style for collecting addresses without extra keystrokes or gates. Instead of the 2006 approach of displaying the address components in a table, our sponsor requested to only display one address at a time. An appropriate form pane had to be chosen so that the FR had a full view of the address items to be edited. Note that the answer list area expands as more addresses are listed or collected.

What was your address on May 1st?

- Probe for complete address including ZIP code.
- Don't include P.O. Box Address.
- To enter Don't Know, press CTRL+D.
- To enter Refused, press CTRL+R.

0. Add new address

1. 100 principal avenue A Nice Place ME

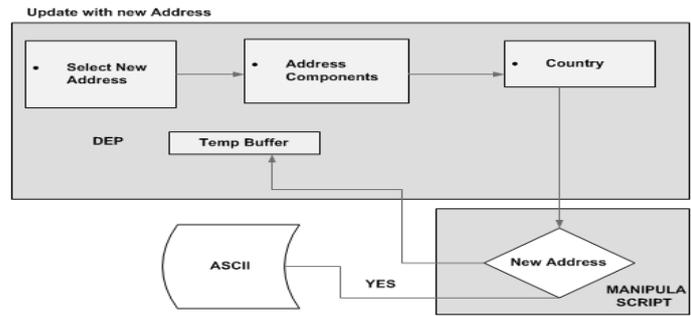
2. 780 main st 201 Happy Place ME

3. 800 point blank 456 Dusty town LA

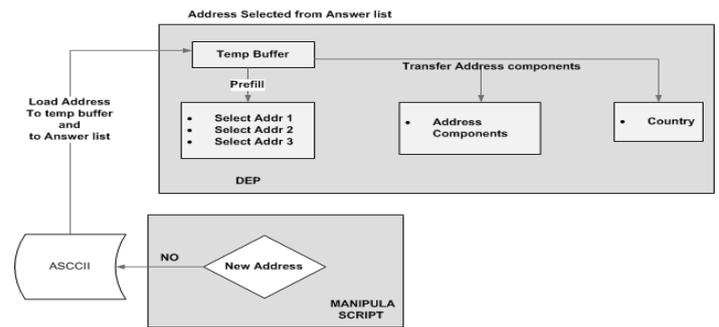
Name	Joe Doe	House#	100	State	ME
HereCD	2	Street Name	principal avenue	Zip	44444
Select	3	Unit	A	Country	1
Probe		City	Nice Place	Mile	
				Cross	RT60 & RT40
				Landmarks	two buildings

The general process of collecting addresses using a Manipula script to store to an external file and to retrieve the address data is as follows:

- ❑ As the FR lands on the “Select Address” question, a Manipula script populates a temp buffer variable from the external ASCII file and sets the answer list displaying the previous addresses collected.
- ❑ As soon as the FR leaves the “Select Address”, “Probe”, or “Country” question, the handler is passed to the Manipula script. The script will acknowledge the request as either “adding a new address” or “selecting an address already displayed”.
- ❑ When “adding a new address”, the script will verify the value from the “Select Address” question and the “Country” question. The Manipula script will compare this address against other addresses in that ASCII file. If there is a match the script does not append a new entry to the external file, otherwise the address components are written out and the temp buffer variable is updated for subsequent residents.



- When selecting an address already displayed, assignments are made from the temp buffer to the address components.



4. Output Processing

Our usual survey sponsors request either the Blaise blobs and conduct their own output processing or they request an ASCII Relational or ASCII output dump and use SAS to process the output data. Our CCM sponsor is not familiar with Blaise blobs, so they preferred a slightly different ASCII output.

4.1 How Output was processed for 2006

In 2006, the instrument output was an extremely long space delimited ASCII file; the dump came out in the order the fields were defined in the metafile. Since the instrument allowed for collection of data for 49 persons and up to 30 addresses in different sections, the output contained spaces reserved for all variables, thus the file totaled more than 1,000,000 characters per case.

Since the VAX machine used in 2006 could not deal with stream output records this long the output was split between variables. Manipula script accomplished this process of splitting it and cameleon was customized to follow the same concept as Manipula in order to produce the SAS script. There were no specified split locations but per case each of the forty-four output records averaged about 30,000 characters in length. The backend team processing this data had to determine where splits were and reorganize the data accordingly. This was obviously very challenging for the data processors to work with.

4.2 How Output is processed for 2009

Based on the output issues encountered back in 2006, a new output method was requested. The ASCII dump option was eliminated since the instrument was even larger for 2009 and because of the difficulty in linking information. The use of an ASCII Relational output was suggested and then eliminated because of the number of files to manage and the extra cleaning to be performed. Also, the current structure of this metafile was too convoluted to make drastic changes. Consequently, creating a customized output script was the only solution to this dilemma.

For the 2009 CCM instrument, the authoring team decided to utilize Blaise 4.8 new capabilities to create a Manipula script to output data in a well known format record typed used by the Census Bureau. One concern with using this approach was that we would need a second customized script to run output for the CCM PI RI instrument.

This approach undoubtedly imposed a concern in maintenance and coordination for any changes done to either instrument or the script to correct data issues.

The script was built from scratch. The following generic guidelines were considered for this ambitious approach:

- ❑ The script must be easy to maintain in the event of new enhancements
- ❑ The script must be able to read a file containing the information about the variables to be written for output and how (this file is also known as the layout file)
- ❑ The script must retrieve for data based on the layout file
- ❑ The script must generate a SAS script and by this we would not use cameleon to obtain information of the metafile since Manipula now is able to read it as well.
- ❑ The script must be generic enough so it can be used for both the CCM PI instrument and the CCM Re-Interview (RI) instrument using a different layout file.

4.2.1 The script must be easy to maintain and allow for new enhancements

The customized script was modularized and broken into several procedures each one doing one specific task and properly parameterized and generic enough to become reusable.

The Procedure below identifies if the variable for output is an array or set of. The “in_search” parameter would indicate what to look for. In this case the presence of the character “[” indicates array and “-” indicates “set of”.

```
PROCEDURE prc_ParseTokenIfSet {third procedure to hit and detect if the token is an arrayed
PARAMETERS
  IMPORT inTokenfld : STRING {token to be read}
  IMPORT inSearch   : STRING {String to search on such as ; - @}
  EXPORT IsFound   : Integer
  EXPORT IndexRow  : STRING
  EXPORT LowRange  : STRING
  EXPORT HiRange   : STRING
  EXPORT HiRange   : STRING
AUXFIELDS
F,P : INTEGER
INSTRUCTIONS
  IF (POSITION(inSearch,inTokenfld)>0) THEN
    IndexRow := SUBSTRING(inTokenfld, (POSITION(inSearch,inTokenfld))-1,
                        (LEN(inTokenfld)-(POSITION(inSearch,inTokenfld)))+1 )
    LowRange := SUBSTRING(IndexRow,1, (POSITION(inSearch,IndexRow))-1)
    HiRange  := SUBSTRING(IndexRow,(POSITION(inSearch,IndexRow))+1,LEN(IndexRow))
    IsFound  := 1
  ELSE
    IndexRow := ''; LowRange := ''; HiRange := ''; IsFound := 2
  ENDIF
ENDPROCEDURE
```

The procedure below formats output values for the various data types. For example, it will correct the format for the nonresponse values, it will right justify numerical variables, and it will left justify alpha variables. This procedure will also check and format datatype or timetype for output.

```

PROCEDURE prc_FormatOutput
{*****}
Created by RPV to format output variable including nonresponse, jul- 2007
{*****}
PARAMETERS
  IMPORT in_variable      : STRING
  EXPORT ex_temformatValue : STRING
AUXFIELDS
  MytempValue, MyVariableName, MyType, MyValue : STRING
  Mysizefld      : INTEGER
  TimeField: TIMETYPE {Added for writing variables associated}
  DateField: DATETYPE {Added for writing variables associated, but not used maybe in the future}

INSTRUCTIONS
  MytempValue := ''; MyVariableName := ''; MyType := ''; MyValue := '';
  Mysizefld:=0 ; TimeField :=EMPTY;DateField :=EMPTY{init variables}
  MyVariableName := in_variable
  MyType := InstrInput.GETFIELDINFO(MyVariableName,'BaseFieldTypeName' )
  MyValue := InstrInput.GETVALUE(MyVariableName,UNFORMATTED)
  Mysizefld := VAL(InstrInput.GETFIELDINFO(MyVariableName, 'SIZE'))
  IF MyType = EMPTY THEN prc_WriteLogFile('ei',MyVariableName) ENDIF
  CASE MyValue OF
    '?' : IF Mysizefld = 1 THEN MytempValue:=REPLACE(MyValue,' ','9');
          ELSE MytempValue:=FORMAT(MytempValue,Mysizefld,RIGHT,'9') ENDIF
    '!' : IF Mysizefld = 1 THEN MytempValue:=REPLACE(MyValue,' ','8');
          ELSE MytempValue:=FORMAT(MytempValue,Mysizefld-1,RIGHT,'9') + '8' ENDIF
  ELSE
    IF MyType = 'STRING' THEN MytempValue:= FORMAT(Myvalue,Mysizefld,LEFT)
    ELSEIF MyType = 'DATETYPE' THEN DateField := STRTODATE(MyValue,MMDDYY)
      MytempValue := SUBSTRING(Myvalue,5,2)+SUBSTRING(MyValue,7,2)+
        SUBSTRING(Myvalue,1,4)
    ELSEIF MyType = 'TIMETYPE' THEN TimeField := STRTOTIME(MyValue)
      MytempValue := FORMAT(STR(TimeField.HOUR),2,RIGHT,'0')+
        ':'+FORMAT(STR(TimeField.MINUTE),2,RIGHT,'0') + ':'+
        FORMAT(STR(TimeField.SECOND),2,RIGHT,'0')
    ELSE
      MytempValue:= FORMAT(Myvalue,Mysizefld,RIGHT)
    ENDIF
  ENDCASE
  ex_temformatValue := MytempValue
ENDPROCEDURE

```

Below we demonstrate the new feature in Blaise 4.8, passing the metafile as parameter (late binding), so we may no longer need to recompile the script; this is a powerful enhancement made to the software.

```

PROCESS ProduceData " **** Writing customize output for CMM **** "

SETTINGS DESCRIPTION = '**** Write Output data and Read Meta information ****'
TIMEFORMAT = HHMMSS {Added for writing variables associated}
DATEFORMAT = MMDDYY {Added for writing variables associated}

USES InputMeta (VAR) { metafile passed as parameter }

DATAMODEL OutputMeta FIELDS MyLine : STRING[3000] ENDMODEL {*** layout of
DATAMODEL InputAscii FIELDS {*** layout file with variables ***}
  Levelline : STRING[20] {Record type read from input}
  Instance : STRING[80] {This main contain the max to loop variable mane.
  VariableName : STRING[100] {Physical location of variable in the instrument.

ENDMODEL.

```

Below we show under the Manipulate section the main entrance to the script. The script can extract data only, can create the SAS script only or if necessary can create both. More important, it is parameterized providing flexibility when running it.

MANIPULATE

```

typeProcess := PARAMETER(1) {'all'}
IF PARAMETER(2) = '' THEN locationInput := 'MyData.inp'
ELSE
    locationInput := PARAMETER (2)
ENDIF
IF PARAMETER(2) = '' THEN param_2 := 'NULL' ELSE param_2 := PARAMETER(2) ENDIF
IF PARAMETER(1) = '' THEN param_1 := 'NULL' ELSE param_1 := PARAMETER(1) ENDIF

prc_clearIt
prc_Init
CASE typeProcess OF
    'data': prc_InitDataDump('Start Custom output Only')
    'dict': prc_InitMetaDictionary('Start Custom Dictionary only')
    'all' : prc_InitDataDump('output and')
           prc_InitMetaDictionary('Dictionary')
ELSE display('No parameters given! or wrong order of parameters passed to this script!
            'This is what the script is accepting as parameters : param(1):'+param_1+' ;
            'Contact Roberto Picha from TMO Authoring if more assistance is required.'
ENDCASE

```

4.2.2 The script must read a file containing the information of the variables to be written for output

Our sponsors provided the team with a dictionary layout of all variables expected in output. The excel document had information such as variable name, length expected and expected values. Authoring added additional information to this excel file - including the physical location of the variable as defined in the metafile and whether or not the variable was “rostered” or “Set of”. This file was then exported to an ASCII layout based on the layout defined for output.

The graphic below demonstrate the excel File layout that eventually will be converted into an ASCII TAB delimited. The last three columns from the excel dictionary were exported; this was a manual process preformed by copying the three columns from excel and pasting to a text editor and then saving it.

Excel Spreadsheet submitted by our sponsor.

A	B	D	E	F	G	H	K	L	M	N
Layout for CCM PI Data Dictionary Layout V1.07-02112009.xls							Record Ouput	Instance	ion in the CCM PI instru	
	Field Name	Start Pos.	End Pos.	Valid Values	Housing Unit, Case, or Person	Character or Numeric Variable	Description			
	PRKM_HM	796	801		H	C	Puerto Rico KM/HM			r2030.PRKM_HM
	PRMUNIC	802	841		H	C	Puerto Rico Municipio			r2030.PRMUNIC
	PRZIP	842	846		H	C	Puerto Rico Zip			r2030.PRZIP
RT 2530	BLOCKNUMBER	847	851		H	C	Block used by CCM			BLOCKNUMBER
	BLOCKSUFFIX	852	853		H	C	Block Suffix used by CCM			BLOCKSUFFIX
SET BY INSTRUMENT				Blanks are valid values for all output since the Listed are valid						N/A
FRONT PI								RecordType8500		
A1	START_A	5	5	1 Continue 2 Ready to transmit 3 Quit,	H	N	First screen. Continue, transmit or end interview attempt			FRONT.START_A
A2	ATTEMPT_TYPE	6	6	1 Personal visit to sample address 2 Personal visit to proxy respondent 3 ...	H	N	Select attempt type or end interview attempt			FRONT.ATTEMPT_TYPE

The resulting ASCII layout file is shown below from the excel dictionary above.

Note that for rostered variables we used the “[]” to indicate script that we are dealing with arrays variables, “Persx” is a variable in the instrument and will instruct the Manipula script to loop as many lines as necessary.

Ascii Layout

```

RecordType8500
      FRONT.START_A
      FRONT.ATTEMPT_TYPE
      FRONT.TRANSMIT
      PRX_OK
      FRONT.PRX_REASON
      FRONT.START_B
      FRONT.NOANSWER

RecordType8502
      Persx TRoster.Person[ ].PERS
      Persx TRosterReview.BPersonReview[ ].ROSTER_REV
      Persx TRoster.Person[ ].DELETE
      Persx TRoster.Person[ ].RESPONDENT
      Persx Demographics.reference[ ]
      Persx TRoster.Person[ ].ROSFLG
      Persx TRoster.Person[ ].FNAME
      Persx TRoster.Person[ ].MINIT

```

4.2.3 The script must retrieve data based on the layout file

Once the layout file was generated from the sponsor's dictionary, the Manipula script retrieved items in the metafile using the GETFIELDINFO and GETTYPEINFO method. This was a critical step inside this script as using these methods allow us to format output for variables that where strings, integers, arrayed or variables defined as Set. Several checks where performed prior to final writing to maintaining data consistency.

The example below of the SCIF format generated by the customized output script.

```

100027000066272827281098AA0001100043027200901 06077G2STXN408203ZE3
850011 0 1 4 1
8502 11011 2MALCOLM DDONEWITH
8502 21000 2JOEL BJUNEBUG
8502 31000 2ISAAC VISLANDER
8502 41000 3ANDREW KHELLFELLOW
8502 51000 5BABY BBOPP
8502 65000 8PAULA KPERKY
8503 131
8503 2 0 102 RIVERSIDE DR SUITE A
8503 331
8503 4 1 102 RIVERSIDE DR SUITE A
8503 531

```

4.2.4 The Script must Generate a SAS script

Since our sponsor's use SAS to process the data we wanted to provide them with the corresponding SAS code to help in their processing. The customized Manipula script was generic enough to expand and enhance further to generate the SAS script in the same manner as it was used when producing data. Again the methods GETFIELDINFO and GETTYPEINFO played a major role here in providing relevant information of each variable listed for output.

The example below shows the result of the SAS script generated by the customized output Manipula script.

Generated SAS Script.

```

data
RT1000 (Keep = CASEID CTRLNUM CASEID CTRLNUM MODE SITE INTPER
RT8500 (Keep = CASEID CTRLNUM START_A ATTEMPT_TYPE TRANSMIT PI
RT8502 (Keep = CASEID CTRLNUM PERS ROSTER_REV DELETE RESPONDEI
RT8503 (Keep = CASEID CTRLNUM PERS ROSTER_ADDR1 ROSTER_PROBE I
RT8504 (Keep = CASEID CTRLNUM PERS ADDR_ID INMVR_ADDR1 INMVR_I
RT8505 (Keep = CASEID CTRLNUM PERS OUTMOV_DATE1 OUT_MONTH OUT_
RT8506 (Keep = CASEID CTRLNUM PERS COLLEGE_ADDR1 COLLEGE_PROBI
RT8507 (Keep = CASEID CTRLNUM PERS SHARED_ADDR1 SHARED_PROBE I
RT8508 (Keep = CASEID CTRLNUM PERS MIL_TIME MIL_STAY MIL_TYPE

```

```

if rt = '8530' then input
  OPERS $ 5 - 6
  WHO_REVIEW_ADDRESS1 $ 7 - 7
  MISLINK $ 8 - 17
  REVIEW_ADDR2 $ 18 - 27
  REVIEW_ADDR3 $ 28 - 60
  REVIEW_ADDR4 $ 61 - 80
  REVIEW_ADDR5 $ 81 - 102
  REVIEW_ADDR6 $ 103 - 104
  REVIEW_ADDR7 $ 105 - 109
  REVIEW_ADDR8 $ 110 - 110
  WHO_REVIEW_ADDRESS3 $ 111 - 174
  WHO_REVIEW_DESCRIP $ 175 - 234
  WHO_REVIEW_MILE $ 235 - 235
  WHO_REVIEW_CROSS $ 236 - 335
  WHO_REVIEW_LNDRMKS $ 336 - 435
  WHO_REVIEW_NEIGHBOR $ 436 - 535
:
if rt = '8530' then output RT8530;

```

4.2.5 The script must be Generic Enough to be used for the CCM PI-RI Instrument.

Since the same data processing team is processing the CCM PI RI output data we wanted to use an identical approach for the Re-Interview instrument. In view of the fact that the two instruments were very similar, the concept to parameterize the script was in order. We were able to use the same script for both instruments and just modify the variable layouts (using the excel sheet) and generate two different SAS scripts. The Output process is as follows:

- Each survey sponsor creates and maintains an Excel file containing the data items they require.
- Authoring adds appropriate instrument variable names and locations to the Excel file.
- ASCII layout file is generated manually from Excel File and delivered to data processors.
- Manipula script is run against the layout file to produce SAS scripts.
- Manipula script uses the ASCII layout file and is run against the consolidated Blaise database to generate customized record typed output

Considering the generic approach of this script, there is a prospective tendency to take advantage of using layout files against new instruments developed by the authoring team. Preliminary testing have assured that the script used for CCMPI can be slightly modified and used as another alternative output option by the Bureau.

5. Roster Collections and Roster Review.

Collecting names of residents with different status can be simple and straightforward, however, spec writers and subject matter people can have a different point of view in collecting rosters. The results can turn into the most unusual way to program rosters.

5.1 How Rostering worked in 2006

Due to the complexity of requirements in 2006, roster collection had to be performed in five separate rosters, each one collecting names of individuals living at a sample address. For each roster category, individuals were assigned a specific status flag stored in the field "RosFlag".

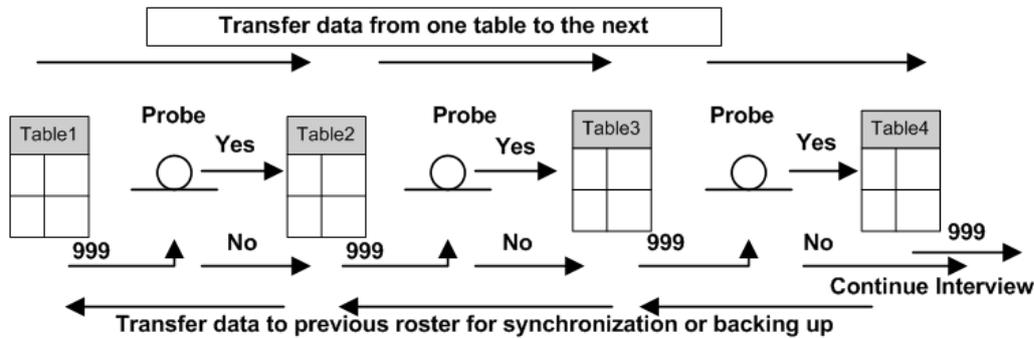
The approach turned out to be convoluted. Though it worked as expected the overhead involved caused some lagging in navigation from question to question. The team suggested that only one roster could have been used for this purpose. However, there was resistance from our sponsors to making changes to the instrument.

The first roster was asked for all individuals who are living at the sample address. Then subsequent probe questions were asked for:

- Individuals who stay at the sample address often,
- Individuals who are staying at the household until they find another place to live,
- Children who were not mentioned yet, and
- Any relatives staying at the sample address.

These five rosters were not combined into one table but they had to be in sync. They remained in separate rosters even in output, which resulted in duplicate data.

The graphic below demonstrates the process of collecting rosters in 2006. This approach required a great deal of code to synchronize all rosters at all times. Flags were required to turn off and on the rules based on the FR manipulation in the Data Entry Program.



5.2 How Rostering worked in 2009 using Blaise 4.8

As part of the new specifications, the 2009 CCM instrument added additional rosters and a final Roster Review where the FR verified, corrected, and modified all names entered in the prior rosters.

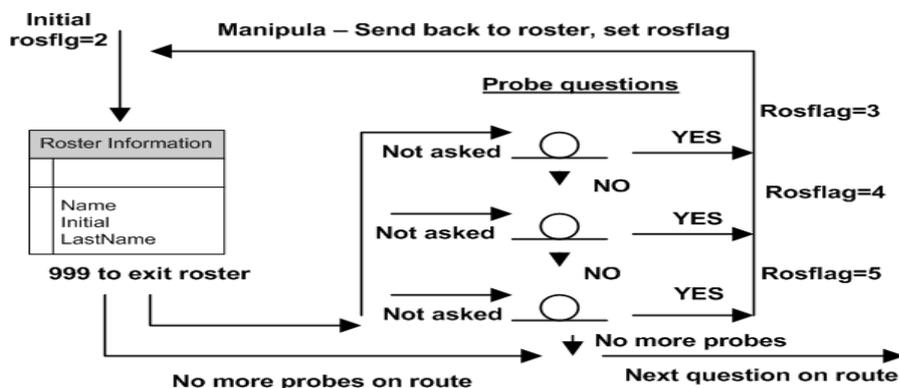
The original code was fairly complex and imposed some issues in situations of backing up to previous rosters. Furthermore, having the rosters synchronized at all times from within the Data Entry Program left an overhead that had to be controlled and simplified.

The authoring team decided that for simplicity there should be only one roster where all names and proper status flag (RosFlag) are stored.

The same roster is also used to store relevant demographics information. This in contrast to the 2006 instrument where data information about individuals was scattered throughout the instrument, making data linkage difficult for analysis.

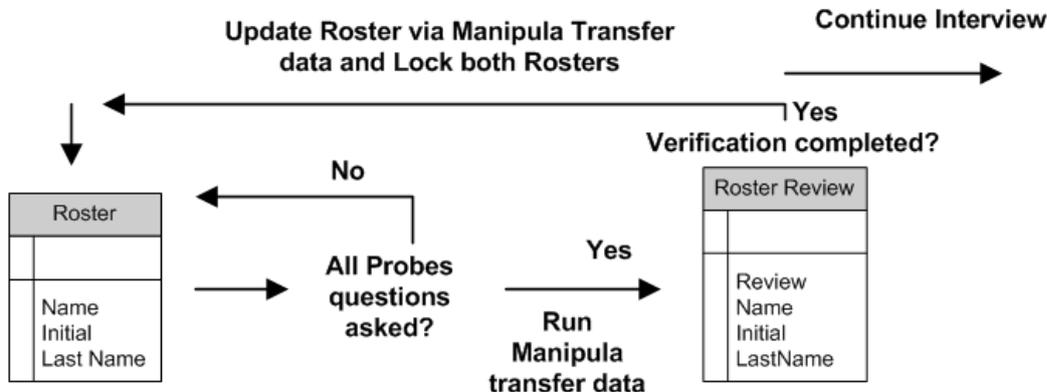
In order to accomplish the goal, the instrument will navigate back to the single roster so it can be updated with missing information when the probe questions are answered with a "yes" response. The "RosFlag" is updated with the appropriate probe code. The probe questions were attached to a Manipula script to handle the setup of "RosFlag" when returning to the roster. Once the table is exited via the 999 token, the instrument asks the subsequent probe question or continues to Roster Review.

The graphic below demonstrates the flow of the Roster Collection with the probe questions flowing back to the roster via to the attached Manipula script.



The graphic below demonstrates the interaction between the Roster and Roster Review. Once the roster is completed via the probes or the "no more people" 999 token, data is transferred to the Roster Review by invoking Manipula through the rules as an alien procedure. The Roster Review process allows the FR to verify, add, and/or update any information that was entered incorrectly. After completing this verification, the updated

information is copied back to the Roster via Manipula. The roster and roster review are then put off path so that the roster information cannot be changed again.



6. Navigating to the Last Question Asked From a Previous Session

One of the new requirements requested by our sponsors for CCM PI 2009 was to have the instrument automatically take the FR to the question they were last on when they previously exited the case (for partial interviews).

Initially the authoring team suggested the use of the “End” key since it is available in the DEP by default; the purpose is similar to the requirement and it is standard function key used in other Census surveys. The “End” key by default goes to the next unanswered question that is not empty enable and meets the universe logic. However the End key does not fully accomplish the request because many of the address components attributes allow for empty and therefore the “End” key could potentially skip over some questions so they may never get asked. Since the CCM interviewers are not experienced FRs, the sponsor believed that this could lead to possible confusion when re-entering partial cases.

In order to accomplish this new requirement, the authoring team used a Manipula script.

The script consists of two procedures and works as follows:

- As soon as the F10 key (exit to back) is pressed, the path location is captured by the Manipula script using the ACTIVEFIELD method and stored into a field called “LastVariable”. “LastVariable” only gets captured when the FR presses the F10 key from the main path of the instrument and the Roster Review is completed. Capturing the last variable does not take place if the user invokes the F10 key from any parallel block in the instrument. The focus is then returned back to the Data Entry Program to resume exit.

```

PROCEDURE prc_LastVariable
INSTRUCTIONS
  IF ACTIVEPARALLEL = 'CCM' THEN
    IF (DEP.Gate_TRoster = 1) OR
      (DEP.WHOutmovers.Gate_TRoster = 1) THEN
      IF (substring(ACTIVEFIELD,1,6) <> 'Front.') AND
        (substring(ACTIVEFIELD,1,5) <> 'Back.') AND
        (substring(ACTIVEFIELD,1,6) <> 'bBack.') THEN
        DEP.LastVariable := ACTIVEFIELD
      ENDIF
    ENDIF
  ELSE
    DEP.LastVariable := ''
  ENDIF
ENDPROCEDURE
  
```

- Once a checkpoint is passed in the re-entered case, Manipula is used to retrieve the content of the “LastVariable” field and sets the focus back to the DEP using the SETACTIVEFIELD function.

```
PROCEDURE prc_SetOnLastVariable
  INSTRUCTIONS
  IF (ROUTERSTATUS= BLRSPREEDIT) THEN
    SETALIENROUTERACTION(BLRAEFAULT)
    SETDATACHANGEDBYUSER(YES)
  ENDIF
  IF (ROUTERSTATUS= BLRSEDIT) THEN
    SETALIENROUTERACTION(BLRAEDITQUESTION)
    SETDATACHANGEDBYUSER(YES)
  ENDIF
  IF (ROUTERSTATUS = BLRSPPOSTEDIT) THEN
    IF DEP.LastVariable <> '' THEN
      ... extra code
      SETACTIVEPARALLEL('CCM')
      SETACTIVEFIELD(DEP.LastVariable)
    ENDIF
    SETALIENROUTERACTION(BLRAEFAULT)
    SETDATACHANGEDBYUSER(YES)
  ENDIF
ENDPROCEDURE
```

7. Conclusion

By using the new capabilities of Blaise 4.8, the TMO Authoring team was able to quickly modify and add enhancements to the CCM PI instrument to meet some of the challenging new requirements for 2009.

We were able to improve the FR/instrument interface for collecting address components to make it smooth, practical, and more efficient by using Manipula with the DEP. The Roster collection and review was made more efficient and cleaner for output purposes. Some user enhancements, such as the navigation to the last question, were also helpful.

However, the most important enhancement for this survey was modifying the way output was created for the data processors. This process not only makes the data processor jobs easier, it also allowed the subject matter specialists to quickly review their data. During early testing, subject matter specialist observed several issues with the output - such as missing variables from the data dictionary and getting extra data that was not expected. These issues were quickly identified and corrected. The new output process was crucial in the success of this project due to the tight turn around with testing the instrument, systems, and reviewing their output data. By quickly identifying data issues, instrument corrections were made in time for later tests and production. The dictionary and layout file seemed to play a big role in defining the dataset expected in output. Also, the size of the output data file per case was significantly reduced compared with the output in 2006, allowing more insightful data analysis.

8. References

Blaise for windows Developer's guide

9. Acknowledgements

The author would like to acknowledge the following people for input into this paper and input work on this project.

Robert Wallace, U.S. Census Bureau
Jason Arata, U.S. Census Bureau
Diane Cronkite, U.S. Census Bureau
Susanne Frank, U.S. Census Bureau
Shadana Myers, U.S. Census Bureau

3. Laying the Foundation

Before starting, we sought input and backing from our Survey Division and senior staff within our Survey Information Services (SIS) group, while consulting with staff within our Computer and Network Services (CaNS) Group and our Computer Assisted Interviewing Support Group (CAISG). As we reviewed our existing systems and programming standards, the advice and assistance from these groups help to determine what we could keep in place and what absolutely needed to change with the migration to 4.8.

We hoped that most of the applications MPR had developed over the years (our overnight processing, interviewer and supervisor CATI menu system, CAPI systems, shell programs for developing instruments, and so on) would not need to change with 4.8; however, we knew the areas of hardware, security, and our network would absolutely need to change.

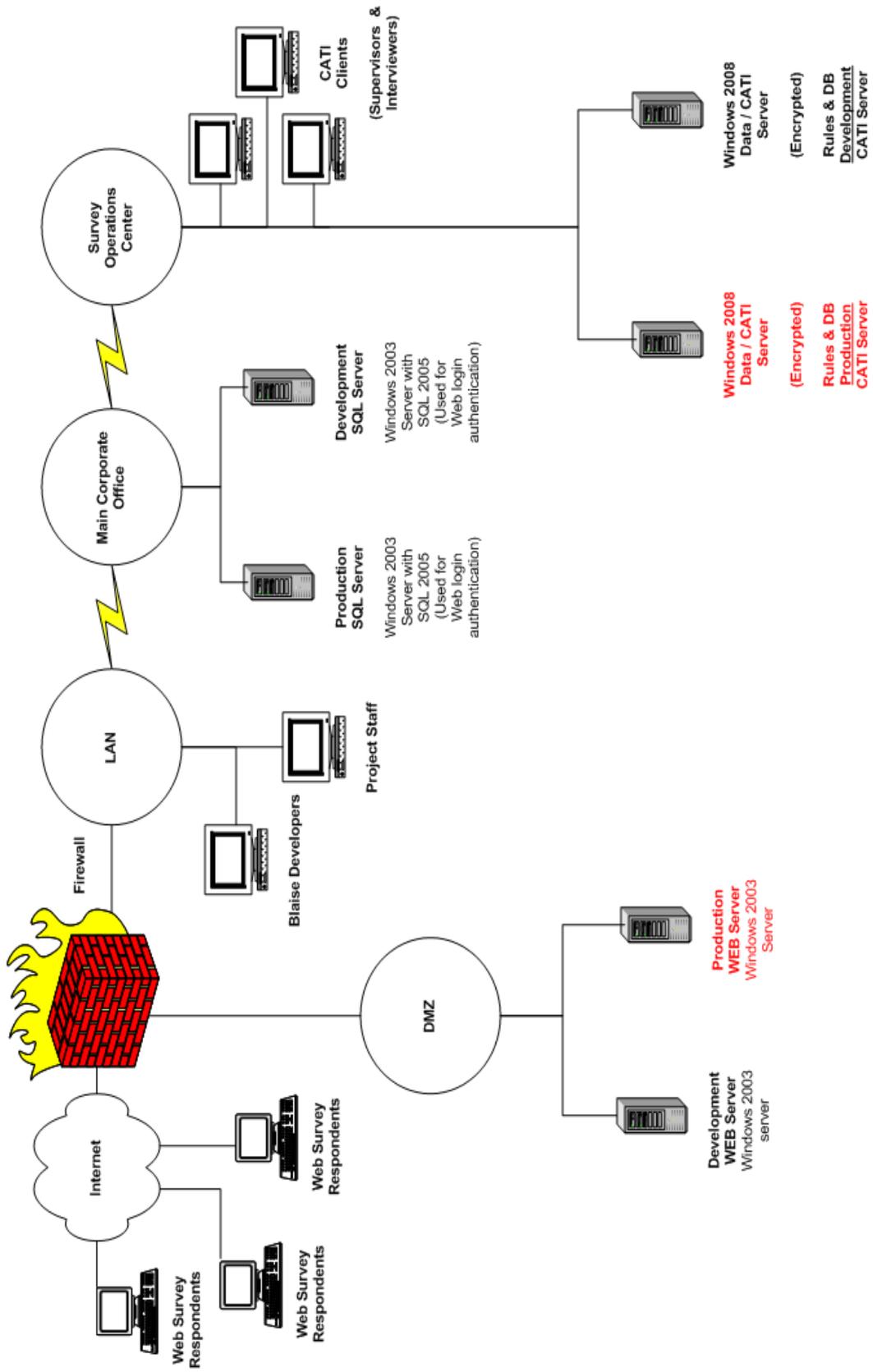
We assembled a team to look at each of these areas and started to put the puzzle pieces in place as we met on a regular basis.

4. Hardware and Infrastructure Changes

The move to Blaise 4.8 presented MPR with an opportunity to upgrade various systems and hardware. Our existing data collection servers had reached the end of their maintenance contracts; the change to Blaise 4.8 helped facilitate a move toward the latest technology and helped us become more consistent on the data collection side with MPR's goal of being a full "Microsoft shop."

Because we were making hardware changes, we wanted to move toward the latest version of Windows Server software available for our data servers, as this would provide us with built-in encryption of data at rest. Any version of Windows Server prior to Windows Server 2008 would require MPR to purchase a third-party application to provide encryption of data at rest. Because we wanted to go to Windows Server 2008, we had some minor concerns about Blaise 4.8 functioning properly as this new operating system had been publicly available for only a few months. Through testing on our development machines running Windows Server 2008 and consultations with SN, we were quickly able to determine Blaise 4.8 would be able to run in the Windows Server 2008 environment and we began to purchase, install, configure, and test the hardware and software necessary to get 4.8 into production.

The figure on the next page shows the current hardware and infrastructure put into place for data collection using Blaise 4.8 in the CAWI and CATI environments at MPR. During the planning and early testing phase, we went through several variations of this as we fine-tuned our system.



5. Testing CAWI and CATI

Stress testing of Blaise Internet instruments—making sure the Blaise 4.8 system could handle our anticipated loads—was a major consideration for MPR. The inability to handle multiple surveys with numerous concurrent users would have been a deal breaker for the company moving forward with Blaise 4.8 into production.

MPR utilized a load-testing tool that simulates real-world internet data collection that requires little human interaction. Web Performance Suite (Web Performance Inc., www.webperformanceinc.com) was the software used to put Blaise 4.8's IS components and our production equipment under stress. After numerous load sessions conducted on our newly implemented infrastructure, as well as on the systems at SN, we were able to replicate up to 250 concurrent internet users reaching a CAWI instrument without putting much stress on our system. Toward the end of our load-testing period, which included 250 concurrent internet sessions while simultaneously having staff perform CATI testing using the same Blaise data set, we were confident we could easily handle more than 250 concurrent internet users within our infrastructure and not have an effect on CATI as well. We felt confident the system was sufficiently robust to handle the anticipated loads our production surveys might encounter and scalable to add new hardware if performance became a problem.

Some of the manual tests we did with Blaise 4.8 targeted the aesthetics and functionality of Blaise instruments for both CAWI and CATI. We tested to make sure CATI functioned within our new infrastructure, while utilizing our existing programs and menu system. These passed our initial testing with minor changes. We also made sure we could program our internet surveys to MPR standards, that we could get screens to appear as they had appeared in other CAWI packages we use at MPR, and that the functionality of our surveys also performed to our standards. Staff members were asked to test internally and externally to MPR utilizing a variety of browsers at various connection speeds. In our browser testing, we found few major deal breakers and we provided SN with our findings. They made any improvements necessary to support as many of the top browsers as possible.

For our final step in this process, we took a test project, programmed it entirely in Blaise 4.8, and tied it into our new FCP internet authentication piece. We again stress tested everything, proving that we could take a survey from authentication to completion without any errors from the instrument or the internet or data servers.

6. The Importance of Meetings

MPR staff met to discuss the pros and cons of moving from version 4.7 to 4.8 before we started to move forward with Blaise 4.8 testing. Discussions were not limited to our CAISG staff members, who work with the Blaise suite on a regular basis; they included our CaNS Group as well as staff from our Survey Division.

Areas examined during these initial meetings included the number and types of changes we expected to make in our normal Blaise programming; expected upgrades to or replacements of existing data and internet servers; how to bring staff up to speed on new features and changes; and the types of changes we would need to make to our overall interviewing infrastructure for CATI, CAPI, and CAWI instruments.

When MPR made the final commitment to Blaise 4.8, we scheduled regular meetings with CAISG and CaNS staff involved in the transition and sent a memorandum to all Survey and Information Technology staff outlining our plans over the next several months during our transition. At initial meetings between CAISG and CaNS, we discussed the plans for the transition, what each group would be responsible for, time lines for acquiring new hardware, testing the systems and software, and ultimately when we would need to be in production.

Our groups met on a weekly basis to discuss the prior week's progress, to raise questions, and to discuss the problems found as we added new pieces to the puzzle. This enabled us to brainstorm how best to leverage the change from 4.7 to 4.8 into other areas involving existing systems or servers that we might not have had the opportunity to change.

As the months progressed and the pieces fell into place, we decided to meet less frequently. Weekly meetings became monthly, until we were approximately five months into production when we decided to meet on a quarterly basis and expand the scope from Blaise-related topics to all things involving computer-assisted interviewing.

Having these regularly scheduled group meetings and making sure to set goals and agendas for each meeting helped smooth the transition and keep everyone involved up to speed and thinking about the overall goal of

getting Blaise 4.8 into production. A side benefit to these regular meetings was that it helped each group involved understand the overall challenges faced by other groups in maintaining our corporate computing infrastructure as well as the challenges of multimode data collection.

In conjunction with the internal meetings with MPR staff, we also had numerous conference calls and email exchanges with several members of the staff at SN. We discussed issues we found in our testing of the Blaise 4.8 system and worked with them to make improvements that we felt would benefit not only MPR, but the Blaise community as a whole.

A tool that was extremely helpful in our meetings with SN was “GoTo Meeting” (<http://www.gotomeeting.com>). Using it, we were able to show SN staff exactly what the issues were, as if they were sitting in our offices. This tool also enabled SN staff to take control of a session if they wanted to dig further into the problem. All parties could quickly discuss issues without having to rely on lengthy email exchanges or phone calls, making for productive use of the limited time we had when both our offices were operating within our normal working hours.

7. Successes

As we finalized this paper, MPR has six instruments in production using Blaise 4.8. Of these six, three have been in production, or recently completed production, over the past nine months. Some of the highlights of these projects follow.

- A project with a sample size of approximately 18,000 potential respondents collected more than 11,000 completed surveys in CATI, CAWI, and CADE modes. Two-thirds of the completed surveys were via CAWI.
- A project with a sample size of approximately 3,300 potential respondents collected slightly more than 1,800 completed surveys in CATI and CAWI modes. The overwhelming majority of completed surveys (approximately 82 percent) were via CAWI.
- A CATI-only project with a sample size of approximately 13,000 collected more than 7,500 completed surveys.

With the conversion of our Blaise CAWI surveys to Blaise IS 4.8.1, along with the FCP login authentication, we have noticed the number of calls and emails to MPR project internet help desks has significantly decreased. Most of the interaction between respondents and those staffing the help desks has been for project-specific issues rather than for the systems- or browser-related issues that seemed to pop up with regularity on the surveys run with iChain and C2B. This has significantly reduced the amount of time CAISG staff spends investigating internet-related issues for projects, saving them money.

8. Issues

Moving to Blaise 4.8.1 has not been all sunshine and happiness, as MPR has experienced its share of “bleeding edge” moments. We expected there would be some bumps in the road along the way and it is impossible to anticipate every possibility when trying out updated software for the first time in a production environment. Some issues we attributed to the learning-curve; others were issues with the software itself that required revisions by SN with new builds MPR needed to put immediately into production.

We found the majority of issues prior to starting projects in production; however, call scheduler issues were found when CATI was involved and have been or are being addressed by SN. The issues that MPR found included: CATI specification file settings were not working; daybatches did not exclude cases as they should have; cases were incorrectly redelivered to interviewers; ToWhom groups were not set properly and cases were delivered to the wrong interviewers; time zone adjustments had been made incorrectly; and interviewers were not being prompted before the next case was automatically delivered by the CATI scheduler. Almost the entire set of internet-related issues presented themselves during our testing and development phases. MPR had a larger concern for Blaise IS 4.8 after we scrapped using previous versions of Blaise IS that were not ready for real-time multimode production use. For that reason we concentrated the large majority of our testing efforts on CAWI rather than CAPI or CATI, where we anticipated finding fewer issues.

Some of the other problems we encountered while in production were loss of survey data due to Blaise application programming interface (Blaise API) and CATI service failures, learning to deploy new instruments carefully (it took time for CAISG staff to get used to waiting for Blaise services to release files or how to work easily around it by “killing” connections made to the database), getting up to speed on Blaise Internet (each new build of Blaise brought with it changes and we were learning on the fly), and we needed to rethink our philosophy on the installation of the Blaise suite to our CAISG developers; previously we had loaded everything a developer needed from our network in one central location, but now we needed to install Blaise 4.8 on individual desktops.

9. Conclusion

Moving to Blaise 4.8 was not without its struggles. The planning we put into place, the testing we did before going into production, following up on issues with SN and Blaise Services at Westat, and the cooperation and understanding of MPR staff were instrumental in successfully getting several surveys up and running without retraining Blaise end users. In the long run we feel MPR will be better positioned to meet the challenging needs of our clients.

We are still experiencing a few problems, but they are becoming less prevalent with each new build of Blaise 4.8. Not everything we anticipated being a major challenge at the beginning of this process turned out to be an issue. For example, before converting to Blaise 4.8 almost all staff at MPR loaded the Blaise applications they needed from a network drive, excluding CAPI surveys. Because the installation of the Blaise suite now follows the Microsoft style of installing all applications locally, we were concerned MPR support staff might have to install the entire Blaise suite on individual machines. This would be a nightmare to maintain for the number of users we have, but SN had already thought about this for several of the Blaise applications and tools. We can still load DEP.EXE or Manipula.EXE from a network drive by installing a set of Microsoft XML dll's on individual users' machines. This enables us to load selected pieces of the Blaise suite from our network. Another example was with the encryption of data at rest. We thought this might be a major task to undertake, but with Windows Server 2008 it turned out to be a very simple task after all.

Two of the major achievements of this transition were being able to rid MPR of older internet technologies and moving toward our corporate goal of being a complete “Microsoft shop.”

Our process of evaluating the tools available in Blaise 4.8.1 and how MPR can leverage them to meet the ever-changing needs of data collection will continue as we build upon our successes. A goal for MPR over the coming months will be to look at adding generic .boi files to instruments which could enable us to centralize survey data into SQL databases and potentially provide us better real-time access to data itself.

Using Blaise 4.8 for Census Coverage Measurement

Roberto V. Picha, U.S. Census Bureau, USA

1. Introduction

This paper discusses a few of the innovations made to the Census Coverage Measurement Person Interview (CCM PI) survey instrument developed by the U.S. Census Bureau using Blaise 4.8. The original instrument was developed in preparation for the 2006 Dress Rehearsal using Blaise 4.6. As the 2010 Decennial Census operations approached, the Technologies Management Office Authoring Staff was requested to program the 2009 CCM PI Dress Rehearsal instrument by starting with the 2006 instrument and adding enhancements to it. The updated instrument specs contained new sets of requirements that presented challenges for the Authoring Staff.

This paper covers how the following functionality was enhanced or added to the 2009 CCM PI Dress Rehearsal instrument:

- ❑ Collecting, Displaying and Selecting Addresses
- ❑ Output Processing
 - Roster Collections and Roster Review
 - Navigating to the Last Question Asked From a Previous Session

2. Background

The CCM PI is conducted by personal visit and telephone using a computer-assisted data collection instrument on a laptop computer. The purpose of the CCM PI is to obtain person and address information about current residents of the sample housing unit, residents who may have moved since Census Day, and residents who may have moved out of the sample housing unit between Census Day and the time the CCM PI interview is conducted. The instrument collects demographic information for each person in the sample housing unit including name, gender, age, date of birth, race, and Hispanic origin. Information is also collected to determine where each current resident was living on Census Day and where each resident who has moved out since Census Day currently lives. Lastly, the instrument also collects information to determine if there are any other addresses where residents may have been counted on Census Day and other information necessary to geocode their alternate addresses. This data will then be compared to the decennial data in order to measure the accuracy of the person count of the 2010 Census.

In addition to developing the CCM PI instrument, a CCM Re-Interview (RI) instrument was created for verifying how the PI was conducted and will actually run the full PI if needed.

The CCM PI survey sample size is extremely large and will be implemented by Field Representatives (FRs) that have been temporarily hired to conduct this interview. Therefore, the premise for some of the CCM PI requirements is that the FRs collecting data do not have the usual experience, understanding, and training of current CAPI FRs. To this end, some of the requirements requested and implemented are ones that are not normally allowed in our current survey instruments.

Examples of these types of requirements are:

- ❑ Allowing empty responses for questions that should be answered
- ❑ Helping the FR with navigation upon re-entry
- ❑ Forcing forward navigation and locking complete sections that would alter the data collection if changed.

3. Collecting, Displaying, and Selecting Addresses

One of the more challenging requirements for the 2006 CCM PI instrument was to collect, display all collected addresses, and allow the FR to select an address from a list of up to 30 addresses per case. Each unique address entered for the various household members was added to an external ASCII file. The contents of this file were presented to the FR in the form of an answer list so the FR could select a previously entered address. When completing the address collection portions of the instrument, the FR either selected one from this list or added a new address. This same list of addresses was constantly updated and kept unique throughout the various sections in the instrument.

In the CCM PI instrument, an address is composed of seven components: house number, street name, unit designation, city, state, zip code, and country. Each unique address can be shared across different types of residencies throughout various sections of the instrument. In general, the FR can select an address previously entered in that section or another section for the same case. If the FR selects “Add New Address” at the Select Address question then all components will be available for editing. Otherwise, if the FR selects a previously entered address from the answer list (which displays all unique addresses entered for the case) the information displayed is transferred to each address component and then left as “show” only.

3.1 How Address Collection was accomplished in 2006

The 2006 CCM instrument attached an alien router written in Delphi that was exclusively built to write the address components collected through the Data Entry Program to an external file. Each unique address was written to an ASCII file by this alien router. In order for the alien router to work properly, the 2006 CCM instrument implemented a concept called gates. The gates were extra questions that triggered some events inside the alien router. The entry and exit gates were placed around the address components. Some times there were two gates one after another; the purpose of these gates was to turn on and off the writing of the external file and avoid infinite writing of the same address.

The addition of these gates presented a challenge to the FR of extra, unnecessary, keystrokes needed when collecting data. This was unacceptable from the survey Methodologists perspective; however, the use of gates was considered unavoidable and left in place for 2006.

The graphic below demonstrates one type of gating utilized. The “Correct” questions locked the address information entered in the components. The locking was accomplished by setting flags in the section to disable the address information from editing.

In order to modify the entries made in any of the address information the FR had to change the value of the “Correct” variable and manually navigate to where the change was needed.

Is the address you selected or entered correct?

1. Yes
 2. No

Name	Select	Probe	House#	Street	Unit	City	State	Zip	US	Description	Landmarks	Correct	Continue
Joe Doe	<input type="checkbox"/>	<input type="checkbox"/>	111	main	21	New ctt	Maine	22344	1			<input type="checkbox"/>	<input type="checkbox"/>
Jane Doe	<input type="checkbox"/>	<input type="checkbox"/>										<input type="checkbox"/>	<input type="checkbox"/>
Jill Doe	<input type="checkbox"/>	<input type="checkbox"/>										<input type="checkbox"/>	<input type="checkbox"/>

The “Continue” question enabled the next row if the next resident had an address in this section; otherwise the FR was taken to the next section.

Name	Select	Probe	House#	Street	Unit	City	State	Zip	US	Description	Landmarks	Correct	Continue
Joe Doe	0		111	main	21	New ctt	Maine	22344	1			1	
Jane Doe													
Jill Doe													

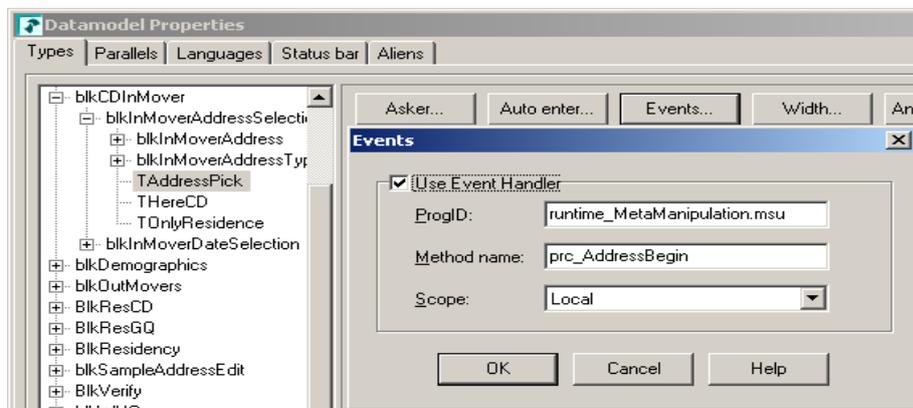
3.2 How Address Collection was implemented in 2009 using Blaise 4.8

For the new 2009 instrument, our sponsor wanted to make address collection more efficient and user-friendly by eliminating the extra questions added for gating the address components. This imposed a series of challenges to the authoring team. First, the original developers that worked with the Delphi application were no longer in the Division and secondly, our team no longer had a copy of this software.

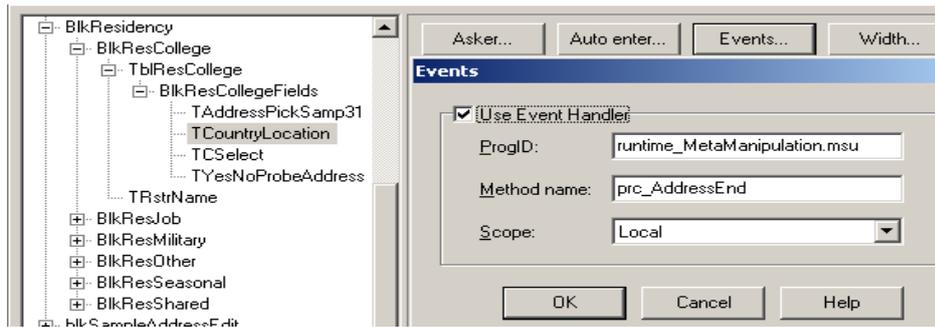
Since Blaise 4.8 offers a robust capability to allow the Data Entry Program to interact with Manipula, a decision was made to migrate the original code written in Delphi to Manipula. This approach proved to be successful and straightforward, obtaining the same results as the original approach plus the additional new requirements requested by our sponsors. They requested that the 2009 CCM PI instrument filter addresses based on the section in the instrument and compare addresses for possible duplicates among other addresses already entered in the various sections of the instrument.

The CCM PI is composed of about twenty-five address sections, each of which collects the same type of address components. This similarity made the implementation of the Manipula script possible. In order to implement the changes, the team removed the four references to the DLL Delphi code and the extra gate fields. We then applied the new call to a Manipula script as an alien procedure. To do this, three questions were mapped to make the call and replace the function of the gates from the original instrument. The first gate was replaced in the “Select Address” question with Datatype TAddressPick, the second in the “probe” question with Datatype TyesNoProbeAddress and the third in the last address component “Country” question with Datatype TCountryLocation.

The graphic below is an example of how to attach to a Manipula script via the data model properties and associate to a data type used in the Answer List for selecting an address.



The same concept was used for writing to an ASCII file in the Country question as shown below.



The following screen demonstrates the new layout and style for collecting addresses without extra keystrokes or gates. Instead of the 2006 approach of displaying the address components in a table, our sponsor requested to only display one address at a time. An appropriate form pane had to be chosen so that the FR had a full view of the address items to be edited. Note that the answer list area expands as more addresses are listed or collected.

What was your address on May 1st?

- Probe for complete address including ZIP code.
- Don't include P.O. Box Address.
- To enter Don't Know, press CTRL+D.
- To enter Refused, press CTRL+R.

0. Add new address

1. 100 principal avenue A Nice Place ME

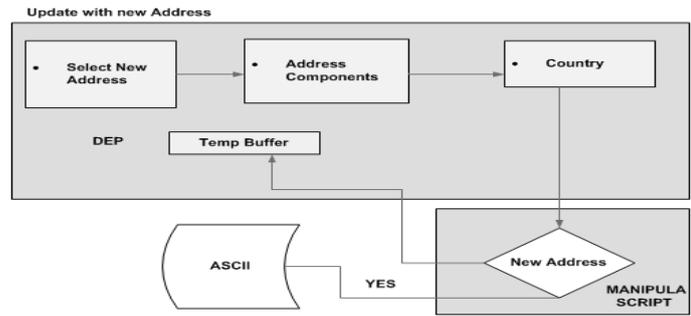
2. 780 main st 201 Happy Place ME

3. 800 point blank 456 Dusty town LA

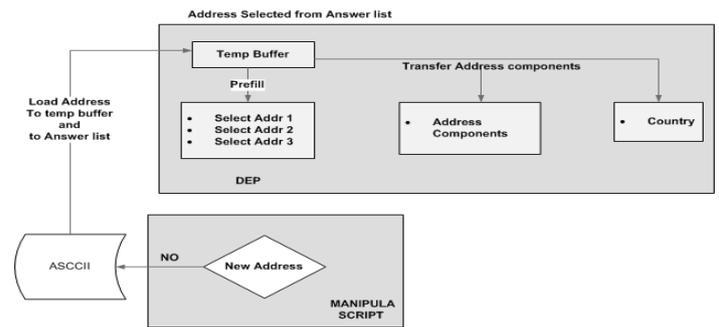
Name	Joe Doe	House#	100	State	ME
HereCD	2	Street Name	principal avenue	Zip	44444
Select	3	Unit	A	Country	1
Probe		City	Nice Place	Mile	
				Cross	RT60 & RT40
				Landmarks	two buildings

The general process of collecting addresses using a Manipula script to store to an external file and to retrieve the address data is as follows:

- ❑ As the FR lands on the “Select Address” question, a Manipula script populates a temp buffer variable from the external ASCII file and sets the answer list displaying the previous addresses collected.
- ❑ As soon as the FR leaves the “Select Address”, “Probe”, or “Country” question, the handler is passed to the Manipula script. The script will acknowledge the request as either “adding a new address” or “selecting an address already displayed”.
- ❑ When “adding a new address”, the script will verify the value from the “Select Address” question and the “Country” question. The Manipula script will compare this address against other addresses in that ASCII file. If there is a match the script does not append a new entry to the external file, otherwise the address components are written out and the temp buffer variable is updated for subsequent residents.



- When selecting an address already displayed, assignments are made from the temp buffer to the address components.



4. Output Processing

Our usual survey sponsors request either the Blaise blobs and conduct their own output processing or they request an ASCII Relational or ASCII output dump and use SAS to process the output data. Our CCM sponsor is not familiar with Blaise blobs, so they preferred a slightly different ASCII output.

4.1 How Output was processed for 2006

In 2006, the instrument output was an extremely long space delimited ASCII file; the dump came out in the order the fields were defined in the metafile. Since the instrument allowed for collection of data for 49 persons and up to 30 addresses in different sections, the output contained spaces reserved for all variables, thus the file totaled more than 1,000,000 characters per case.

Since the VAX machine used in 2006 could not deal with stream output records this long the output was split between variables. Manipula script accomplished this process of splitting it and cameleon was customized to follow the same concept as Manipula in order to produce the SAS script. There were no specified split locations but per case each of the forty-four output records averaged about 30,000 characters in length. The backend team processing this data had to determine where splits were and reorganize the data accordingly. This was obviously very challenging for the data processors to work with.

4.2 How Output is processed for 2009

Based on the output issues encountered back in 2006, a new output method was requested. The ASCII dump option was eliminated since the instrument was even larger for 2009 and because of the difficulty in linking information. The use of an ASCII Relational output was suggested and then eliminated because of the number of files to manage and the extra cleaning to be performed. Also, the current structure of this metafile was too convoluted to make drastic changes. Consequently, creating a customized output script was the only solution to this dilemma.

For the 2009 CCM instrument, the authoring team decided to utilize Blaise 4.8 new capabilities to create a Manipula script to output data in a well known format record typed used by the Census Bureau. One concern with using this approach was that we would need a second customized script to run output for the CCM PI RI instrument.

This approach undoubtedly imposed a concern in maintenance and coordination for any changes done to either instrument or the script to correct data issues.

The script was built from scratch. The following generic guidelines were considered for this ambitious approach:

- ❑ The script must be easy to maintain in the event of new enhancements
- ❑ The script must be able to read a file containing the information about the variables to be written for output and how (this file is also known as the layout file)
- ❑ The script must retrieve for data based on the layout file
- ❑ The script must generate a SAS script and by this we would not use cameleon to obtain information of the metafile since Manipula now is able to read it as well.
- ❑ The script must be generic enough so it can be used for both the CCM PI instrument and the CCM Re-Interview (RI) instrument using a different layout file.

4.2.1 The script must be easy to maintain and allow for new enhancements

The customized script was modularized and broken into several procedures each one doing one specific task and properly parameterized and generic enough to become reusable.

The Procedure below identifies if the variable for output is an array or set of. The “in_search” parameter would indicate what to look for. In this case the presence of the character “[” indicates array and “-” indicates “set of”.

```
PROCEDURE prc_ParseTokenIfSet {third procedure to hit and detect if the token is an arrayed
PARAMETERS
  IMPORT inTokenfld : STRING {token to be read}
  IMPORT inSearch   : STRING {String to search on such as ; - @}
  EXPORT IsFound   : Integer
  EXPORT IndexRow  : STRING
  EXPORT LowRange  : STRING
  EXPORT HiRange   : STRING
  EXPORT HiRange   : STRING
AUXFIELDS
F,P : INTEGER
INSTRUCTIONS
  IF (POSITION(inSearch,inTokenfld)>0) THEN
    IndexRow := SUBSTRING(inTokenfld, (POSITION(inSearch,inTokenfld))-1,
                          (LEN(inTokenfld)-(POSITION(inSearch,inTokenfld)))+1 )
    LowRange := SUBSTRING(IndexRow,1, (POSITION(inSearch,IndexRow))-1)
    HiRange  := SUBSTRING(IndexRow,(POSITION(inSearch,IndexRow))+1, len(IndexRow))
    IsFound  := 1
  ELSE
    IndexRow := ''; LowRange := ''; HiRange := ''; IsFound := 2
  ENDIF
ENDPROCEDURE
```

The procedure below formats output values for the various data types. For example, it will correct the format for the nonresponse values, it will right justify numerical variables, and it will left justify alpha variables. This procedure will also check and format datatype or timetype for output.

```

PROCEDURE prc_FormatOutput
{*****}
Created by RPV to format output variable including nonresponse, jul- 2007
{*****}
PARAMETERS
  IMPORT in_variable      : STRING
  EXPORT ex_temformatValue : STRING
AUXFIELDS
  MytempValue, MyVariableName, MyType, MyValue : STRING
  Mysizefld      : INTEGER
  TimeField: TIMETYPE {Added for writing variables associated}
  DateField: DATETYPE {Added for writing variables associated, but not used maybe in the future}

INSTRUCTIONS
  MytempValue := ''; MyVariableName := ''; MyType := ''; MyValue := '';
  Mysizefld:=0 ; TimeField :=EMPTY;DateField :=EMPTY{init variables}
  MyVariableName := in_variable
  MyType := InstrInput.GETFIELDINFO(MyVariableName,'BaseFieldTypeName' )
  MyValue := InstrInput.GETVALUE(MyVariableName,UNFORMATTED)
  Mysizefld := VAL(InstrInput.GETFIELDINFO(MyVariableName, 'SIZE'))
  IF MyType = EMPTY THEN prc_WriteLogFile('ei',MyVariableName) ENDIF
  CASE MyValue OF
    '?' : IF Mysizefld = 1 THEN MytempValue:=REPLACE(MyValue,'?', '9');
          ELSE MytempValue:=FORMAT(MytempValue, Mysizefld, RIGHT, '9') ENDIF
    '!' : IF Mysizefld = 1 THEN MytempValue:=REPLACE(MyValue,'!', '8');
          ELSE MytempValue:=FORMAT(MytempValue, Mysizefld-1, RIGHT, '9') + '8' ENDIF
  ELSE
    IF MyType = 'STRING' THEN MytempValue:= FORMAT(Myvalue, Mysizefld, LEFT)
    ELSEIF MyType = 'DATETYPE' THEN DateField := STRTODATE(MyValue, MMDDYY)
      MytempValue := SUBSTRING(Myvalue, 5, 2)+SUBSTRING(MyValue, 7, 2)+
        SUBSTRING(Myvalue, 1, 4)
    ELSEIF MyType = 'TIMETYPE' THEN TimeField := STRTOTIME(MyValue)
      MytempValue := FORMAT(STR(TimeField.HOUR), 2, RIGHT, '0')+
        ':'+FORMAT(STR(TimeField.MINUTE), 2, RIGHT, '0') + ':'+
        FORMAT(STR(TimeField.SECOND), 2, RIGHT, '0')
    ELSE
      MytempValue:= FORMAT(Myvalue, Mysizefld, RIGHT)
    ENDIF
  ENDCASE
  ex_temformatValue := MytempValue
ENDPROCEDURE

```

Below we demonstrate the new feature in Blaise 4.8, passing the metafile as parameter (late binding), so we may no longer need to recompile the script; this is a powerful enhancement made to the software.

```

PROCESS ProduceData " **** Writing customize output for CMM **** "

SETTINGS DESCRIPTION = '**** Write Output data and Read Meta information ****'
TIMEFORMAT = HHMMSS {Added for writing variables associated}
DATEFORMAT = MMDDYY {Added for writing variables associated}

USES InputMeta (VAR) { metafile passed as parameter }

DATAMODEL OutputMeta FIELDS MyLine : STRING[3000] ENDMODEL {*** layout of
DATAMODEL InputAscii FIELDS {*** layout file with variables ***}
  Levelline : STRING[20] {Record type read from input}
  Instance : STRING[80] {This main contain the max to loop variable mane.
  VariableName : STRING[100] {Physical location of variable in the instrument.

ENDMODEL.

```

Below we show under the Manipulate section the main entrance to the script. The script can extract data only, can create the SAS script only or if necessary can create both. More important, it is parameterized providing flexibility when running it.

MANIPULATE

```

typeProcess := PARAMETER(1) {'all'}
IF PARAMETER(2) = '' THEN locationInput := 'MyData.inp'
ELSE
    locationInput := PARAMETER (2)
ENDIF
IF PARAMETER(2) = '' THEN param_2 := 'NULL' ELSE param_2 := PARAMETER(2) ENDIF
IF PARAMETER(1) = '' THEN param_1 := 'NULL' ELSE param_1 := PARAMETER(1) ENDIF

prc_clearIt
prc_Init
CASE typeProcess OF
    'data': prc_InitDataDump('Start Custom output Only')
    'dict': prc_InitMetaDictionary('Start Custom Dictionary only')
    'all' : prc_InitDataDump('output and')
           prc_InitMetaDictionary('Dictionary')
ELSE display('No parameters given! or wrong order of parameters passed to this script!
           'This is what the script is accepting as parameters : param(1):'+param_1+' ;
           'Contact Roberto Picha from TMO Authoring if more assistance is required.')
ENDCASE

```

4.2.2 The script must read a file containing the information of the variables to be written for output

Our sponsors provided the team with a dictionary layout of all variables expected in output. The excel document had information such as variable name, length expected and expected values. Authoring added additional information to this excel file - including the physical location of the variable as defined in the metafile and whether or not the variable was “rostered” or “Set of”. This file was then exported to an ASCII layout based on the layout defined for output.

The graphic below demonstrate the excel File layout that eventually will be converted into an ASCII TAB delimited. The last three columns from the excel dictionary were exported; this was a manual process preformed by copying the three columns from excel and pasting to a text editor and then saving it.

Excel Spreadsheet submitted by our sponsor.

A	B	D	E	F	G	H	K	L	M	N
Layout for CCM PI Data Dictionary Layout V1.07-02112009.xls							Record Ouput	Instance	ion in the CCM PI instru	
	Field Name	Start Pos.	End Pos.	Valid Values	Housing Unit, Case, or Person	Character or Numeric Variable	Description			
	PRKM_HM	796	801		H	C	Puerto Rico KM/HM			r2030.PRKM_HM
	PRMUNIC	802	841		H	C	Puerto Rico Municipio			r2030.PRMUNIC
	PRZIP	842	846		H	C	Puerto Rico Zip			r2030.PRZIP
RT 2530	BLOCKNUMBER	847	851		H	C	Block used by CCM			BLOCKNUMBER
	BLOCKSUFFIX	852	853		H	C	Block Suffix used by CCM			BLOCKSUFFIX
SET BY INSTRUMENT				Blanks are valid values for all output since the Listed are valid						N/A
FRONT PI								RecordType8500		
A1	START_A	5	5	1 Continue 2 Ready to transmit 3 Quit,	H	N	First screen. Continue, transmit or end interview attempt			FRONT.START_A
A2	ATTEMPT_TYPE	6	6	1 Personal visit to sample address 2 Personal visit to proxy respondent 3 ...	H	N	Select attempt type or end interview attempt			FRONT.ATTEMPT_TYPE

The resulting ASCII layout file is shown below from the excel dictionary above.

Note that for rostered variables we used the “[]” to indicate script that we are dealing with arrays variables, “Persx” is a variable in the instrument and will instruct the Manipula script to loop as many lines as necessary.

Ascii Layout

```

RecordType8500
      FRONT.START_A
      FRONT.ATTEMPT_TYPE
      FRONT.TRANSMIT
      PRX_OK
      FRONT.PRX_REASON
      FRONT.START_B
      FRONT.NOANSWER

RecordType8502
      Persx TRoster.Person[ ].PERS
      Persx TRosterReview.BPersonReview[ ].ROSTER_REV
      Persx TRoster.Person[ ].DELETE
      Persx TRoster.Person[ ].RESPONDENT
      Persx Demographics.reference[ ]
      Persx TRoster.Person[ ].ROSFLG
      Persx TRoster.Person[ ].FNAME
      Persx TRoster.Person[ ].MINIT

```

4.2.3 The script must retrieve data based on the layout file

Once the layout file was generated from the sponsor's dictionary, the Manipula script retrieved items in the metafile using the GETFIELDINFO and GETTYPEINFO method. This was a critical step inside this script as using these methods allow us to format output for variables that where strings, integers, arrayed or variables defined as Set. Several checks where performed prior to final writing to maintaining data consistency.

The example below of the SCIF format generated by the customized output script.

```

100027000066272827281098AA0001100043027200901 06077G2STXN408203ZE3
850011 0 1 4 1
8502 11011 2MALCOLM DDONEWITH
8502 21000 2JOEL BJUNEBUG
8502 31000 2ISAAC VISLANDER
8502 41000 3ANDREW KHELLFELLOW
8502 51000 5BABY BBOPP
8502 65000 8PAULA KPERKY
8503 131
8503 2 0 102 RIVERSIDE DR SUITE A
8503 331
8503 4 1 102 RIVERSIDE DR SUITE A
8503 531

```

4.2.4 The Script must Generate a SAS script

Since our sponsor's use SAS to process the data we wanted to provide them with the corresponding SAS code to help in their processing. The customized Manipula script was generic enough to expand and enhance further to generate the SAS script in the same manner as it was used when producing data. Again the methods GETFIELDINFO and GETTYPEINFO played a major role here in providing relevant information of each variable listed for output.

The example below shows the result of the SAS script generated by the customized output Manipula script.

Generated SAS Script.

```

data
RT1000 (Keep = CASEID CTRLNUM CASEID CTRLNUM MODE SITE INTPER
RT8500 (Keep = CASEID CTRLNUM START_A ATTEMPT_TYPE TRANSMIT PI
RT8502 (Keep = CASEID CTRLNUM PERS ROSTER_REV DELETE RESPONDEI
RT8503 (Keep = CASEID CTRLNUM PERS ROSTER_ADDR1 ROSTER_PROBE I
RT8504 (Keep = CASEID CTRLNUM PERS ADDR_ID INMVR_ADDR1 INMVR_I
RT8505 (Keep = CASEID CTRLNUM PERS OUTMOV_DATE1 OUT_MONTH OUT_
RT8506 (Keep = CASEID CTRLNUM PERS COLLEGE_ADDR1 COLLEGE_PROBI
RT8507 (Keep = CASEID CTRLNUM PERS SHARED_ADDR1 SHARED_PROBE I
RT8508 (Keep = CASEID CTRLNUM PERS MIL_TIME MIL_STAY MIL_TYPE

```

```

if rt = '8530' then input
  OPERS $ 5 - 6
  WHO_REVIEW_ADDRESS1 $ 7 - 7
  MISLINK $ 8 - 17
  REVIEW_ADDR2 $ 18 - 27
  REVIEW_ADDR3 $ 28 - 60
  REVIEW_ADDR4 $ 61 - 80
  REVIEW_ADDR5 $ 81 - 102
  REVIEW_ADDR6 $ 103 - 104
  REVIEW_ADDR7 $ 105 - 109
  REVIEW_ADDR8 $ 110 - 110
  WHO_REVIEW_ADDRESS3 $ 111 - 174
  WHO_REVIEW_DESCRIP $ 175 - 234
  WHO_REVIEW_MILE $ 235 - 235
  WHO_REVIEW_CROSS $ 236 - 335
  WHO_REVIEW_LNDRMKS $ 336 - 435
  WHO_REVIEW_NEIGHBOR $ 436 - 535
:
if rt = '8530' then output RT8530;

```

4.2.5 The script must be Generic Enough to be used for the CCM PI-RI Instrument.

Since the same data processing team is processing the CCM PI RI output data we wanted to use an identical approach for the Re-Interview instrument. In view of the fact that the two instruments were very similar, the concept to parameterize the script was in order. We were able to use the same script for both instruments and just modify the variable layouts (using the excel sheet) and generate two different SAS scripts. The Output process is as follows:

- Each survey sponsor creates and maintains an Excel file containing the data items they require.
- Authoring adds appropriate instrument variable names and locations to the Excel file.
- ASCII layout file is generated manually from Excel File and delivered to data processors.
- Manipula script is run against the layout file to produce SAS scripts.
- Manipula script uses the ASCII layout file and is run against the consolidated Blaise database to generate customized record typed output

Considering the generic approach of this script, there is a prospective tendency to take advantage of using layout files against new instruments developed by the authoring team. Preliminary testing have assured that the script used for CCMPI can be slightly modified and used as another alternative output option by the Bureau.

5. Roster Collections and Roster Review.

Collecting names of residents with different status can be simple and straightforward, however, spec writers and subject matter people can have a different point of view in collecting rosters. The results can turn into the most unusual way to program rosters.

5.1 How Rostering worked in 2006

Due to the complexity of requirements in 2006, roster collection had to be performed in five separate rosters, each one collecting names of individuals living at a sample address. For each roster category, individuals were assigned a specific status flag stored in the field "RosFlag".

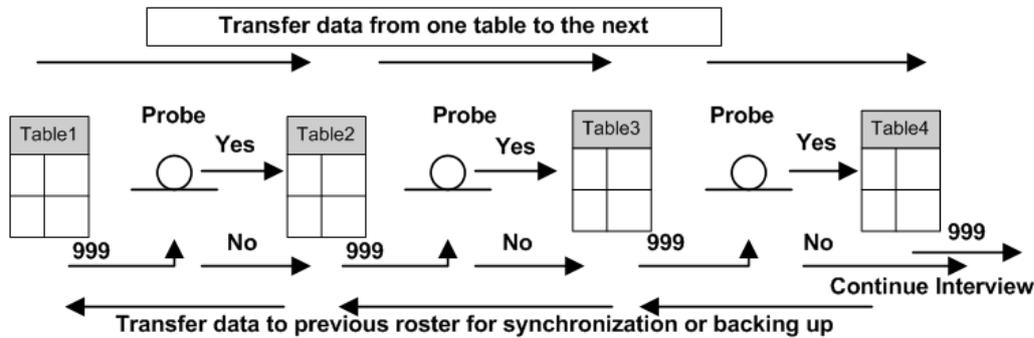
The approach turned out to be convoluted. Though it worked as expected the overhead involved caused some lagging in navigation from question to question. The team suggested that only one roster could have been used for this purpose. However, there was resistance from our sponsors to making changes to the instrument.

The first roster was asked for all individuals who are living at the sample address. Then subsequent probe questions were asked for:

- Individuals who stay at the sample address often,
- Individuals who are staying at the household until they find another place to live,
- Children who were not mentioned yet, and
- Any relatives staying at the sample address.

These five rosters were not combined into one table but they had to be in sync. They remained in separate rosters even in output, which resulted in duplicate data.

The graphic below demonstrates the process of collecting rosters in 2006. This approach required a great deal of code to synchronize all rosters at all times. Flags were required to turn off and on the rules based on the FR manipulation in the Data Entry Program.



5.2 How Rostering worked in 2009 using Blaise 4.8

As part of the new specifications, the 2009 CCM instrument added additional rosters and a final Roster Review where the FR verified, corrected, and modified all names entered in the prior rosters.

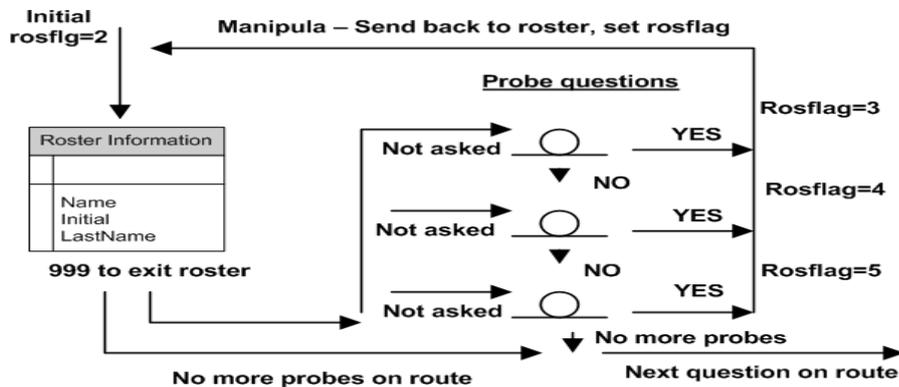
The original code was fairly complex and imposed some issues in situations of backing up to previous rosters. Furthermore, having the rosters synchronized at all times from within the Data Entry Program left an overhead that had to be controlled and simplified.

The authoring team decided that for simplicity there should be only one roster where all names and proper status flag (RosFlag) are stored.

The same roster is also used to store relevant demographics information. This in contrast to the 2006 instrument where data information about individuals was scattered throughout the instrument, making data linkage difficult for analysis.

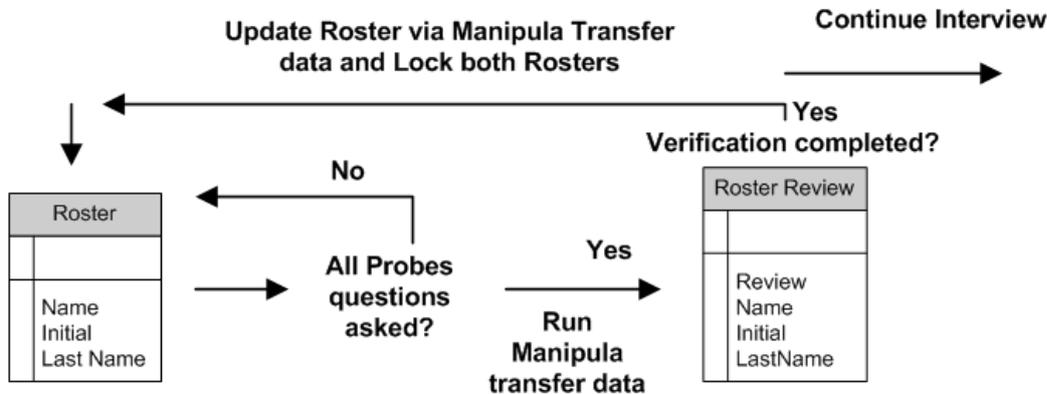
In order to accomplish the goal, the instrument will navigate back to the single roster so it can be updated with missing information when the probe questions are answered with a "yes" response. The "RosFlag" is updated with the appropriate probe code. The probe questions were attached to a Manipula script to handle the setup of "RosFlag" when returning to the roster. Once the table is exited via the 999 token, the instrument asks the subsequent probe question or continues to Roster Review.

The graphic below demonstrates the flow of the Roster Collection with the probe questions flowing back to the roster via to the attached Manipula script.



The graphic below demonstrates the interaction between the Roster and Roster Review. Once the roster is completed via the probes or the "no more people" 999 token, data is transferred to the Roster Review by invoking Manipula through the rules as an alien procedure. The Roster Review process allows the FR to verify, add, and/or update any information that was entered incorrectly. After completing this verification, the updated

information is copied back to the Roster via Manipula. The roster and roster review are then put off path so that the roster information cannot be changed again.



6. Navigating to the Last Question Asked From a Previous Session

One of the new requirements requested by our sponsors for CCM PI 2009 was to have the instrument automatically take the FR to the question they were last on when they previously exited the case (for partial interviews).

Initially the authoring team suggested the use of the “End” key since it is available in the DEP by default; the purpose is similar to the requirement and it is standard function key used in other Census surveys. The “End” key by default goes to the next unanswered question that is not empty enable and meets the universe logic. However the End key does not fully accomplish the request because many of the address components attributes allow for empty and therefore the “End” key could potentially skip over some questions so they may never get asked. Since the CCM interviewers are not experienced FRs, the sponsor believed that this could lead to possible confusion when re-entering partial cases.

In order to accomplish this new requirement, the authoring team used a Manipula script.

The script consists of two procedures and works as follows:

- As soon as the F10 key (exit to back) is pressed, the path location is captured by the Manipula script using the ACTIVEFIELD method and stored into a field called “LastVariable”. “LastVariable” only gets captured when the FR presses the F10 key from the main path of the instrument and the Roster Review is completed. Capturing the last variable does not take place if the user invokes the F10 key from any parallel block in the instrument. The focus is then returned back to the Data Entry Program to resume exit.

```

PROCEDURE prc_LastVariable
INSTRUCTIONS
  IF ACTIVEPARALLEL = 'CCM' THEN
    IF (DEP.Gate_TRoster = 1) OR
      (DEP.WHOutmovers.Gate_TRoster = 1) THEN
      IF (substring(ACTIVEFIELD,1,6) <> 'Front.') AND
        (substring(ACTIVEFIELD,1,5) <> 'Back.') AND
        (substring(ACTIVEFIELD,1,6) <> 'bBack.') THEN
        DEP.LastVariable := ACTIVEFIELD
      ENDIF
    ENDIF
  ELSE
    DEP.LastVariable := ''
  ENDIF
ENDPROCEDURE
  
```

- Once a checkpoint is passed in the re-entered case, Manipula is used to retrieve the content of the “LastVariable” field and sets the focus back to the DEP using the SETACTIVEFIELD function.

```
PROCEDURE prc_SetOnLastVariable
  INSTRUCTIONS
  IF (ROUTERSTATUS= BLRSPREEDIT) THEN
    SETALIENROUTERACTION(BLRAEFAULT)
    SETDATACHANGEDBYUSER(YES)
  ENDIF
  IF (ROUTERSTATUS= BLRSEDIT) THEN
    SETALIENROUTERACTION(BLRAEDITQUESTION)
    SETDATACHANGEDBYUSER(YES)
  ENDIF
  IF (ROUTERSTATUS = BLRSPPOSTEDIT) THEN
    IF DEP.LastVariable <> '' THEN
      ... extra code
      SETACTIVEPARALLEL('CCM')
      SETACTIVEFIELD(DEP.LastVariable)
    ENDIF
    SETALIENROUTERACTION(BLRAEFAULT)
    SETDATACHANGEDBYUSER(YES)
  ENDIF
ENDPROCEDURE
```

7. Conclusion

By using the new capabilities of Blaise 4.8, the TMO Authoring team was able to quickly modify and add enhancements to the CCM PI instrument to meet some of the challenging new requirements for 2009.

We were able to improve the FR/instrument interface for collecting address components to make it smooth, practical, and more efficient by using Manipula with the DEP. The Roster collection and review was made more efficient and cleaner for output purposes. Some user enhancements, such as the navigation to the last question, were also helpful.

However, the most important enhancement for this survey was modifying the way output was created for the data processors. This process not only makes the data processor jobs easier, it also allowed the subject matter specialists to quickly review their data. During early testing, subject matter specialist observed several issues with the output - such as missing variables from the data dictionary and getting extra data that was not expected. These issues were quickly identified and corrected. The new output process was crucial in the success of this project due to the tight turn around with testing the instrument, systems, and reviewing their output data. By quickly identifying data issues, instrument corrections were made in time for later tests and production. The dictionary and layout file seemed to play a big role in defining the dataset expected in output. Also, the size of the output data file per case was significantly reduced compared with the output in 2006, allowing more insightful data analysis.

8. References

Blaise for windows Developer's guide

9. Acknowledgements

The author would like to acknowledge the following people for input into this paper and input work on this project.

Robert Wallace, U.S. Census Bureau
Jason Arata, U.S. Census Bureau
Diane Cronkite, U.S. Census Bureau
Susanne Frank, U.S. Census Bureau
Shadana Myers, U.S. Census Bureau

Using the Blaise Component Pack for All Stages of Data Collection

Lilia Filippenko, Roger Osborn, Vorapraanee (Mai) Wickelgren, & Venkat Yetukuri, RTI International, USA

1. Introduction

At RTI International we have a number of studies where the Blaise Component Pack (BCP) has been used to create alien routers to conduct external assessments during computer-assisted personal interviews (CAPI). In addition, a number of applications have been developed to process Blaise interview data through the entire survey lifecycle.

This paper presents our experience with the BCP in collecting and processing the data for a recent project. During the development stage of the project, we created a number of applications in Visual Basic 6 and .Net to address our client's requirements:

- **Screener Selection** – An alien router that runs a series of algorithms to randomize and select eligible children to participate in a CAPI interview.
- **Create Cases** – An application that spawns a new case on a laptop to conduct an additional interview immediately upon completion of the screener interview. It is also used to create additional interviews during a nightly job at RTI after receiving the completed screener interview.
- **Fingertick Timer** – An alien router and application that helps collect three blood samples during the CAPI interview at predefined time intervals.
- **Bio-Tracking System** – A web-based application to track events associated with bloodspots and saliva collection, shipment, and receipt.
- **Generate Mandatory Report** – An application that flags cases based on responses to certain questions during the CAPI interview.

Due to the complexity of requirements and time constraints, our goal was to have all applications easily configurable and testable. The paper describes developed applications and alien routers. It also includes examples of .Net classes used to monitor calls to executables and log discovered problems or errors.

2. Screener Selection

Many studies require collection of a household roster as part of a screener interview, followed by a CAPI interview with randomly selected household members. The study in question includes a screener interview, which utilizes a large amount of preload data collected from previous waves of this longitudinal study about respondents' children. Respondents have an opportunity to update as well as verify and/or add information regarding their offspring and/or other children they have parented. Upon completion of collecting this information, a series of selection algorithms should select eligible children to participate in the CAPI interview.

These selection algorithms include sorting of rostered children into three groups and using age and/or randomization to select up to two children in a group. Although sorting and randomization could be performed in the Blaise instrument itself, it is a complicated task that requires many hours for testing. We decided to create an alien router to perform the task and simplify the testing by using debugging tools in Visual Basic.

In the Blaise instrument at the top level of the data model we have created three arrays to save information about children:

- **ChildArray[1..13]** – has information needed for selection algorithms about all reported children. This array is used by the alien router to run the selection and is populated with the data just before the call to the alien router.
- **OutChildArray[1..13]** – has additional information about eligibility and results of the selection. This array is populated by the alien router after the selection is performed and is used later for analysis. Records are sorted according to selection algorithms.
- **SelectedChildren[1..2]** – has information about selected child/children needed to spawn CAPI interviews.

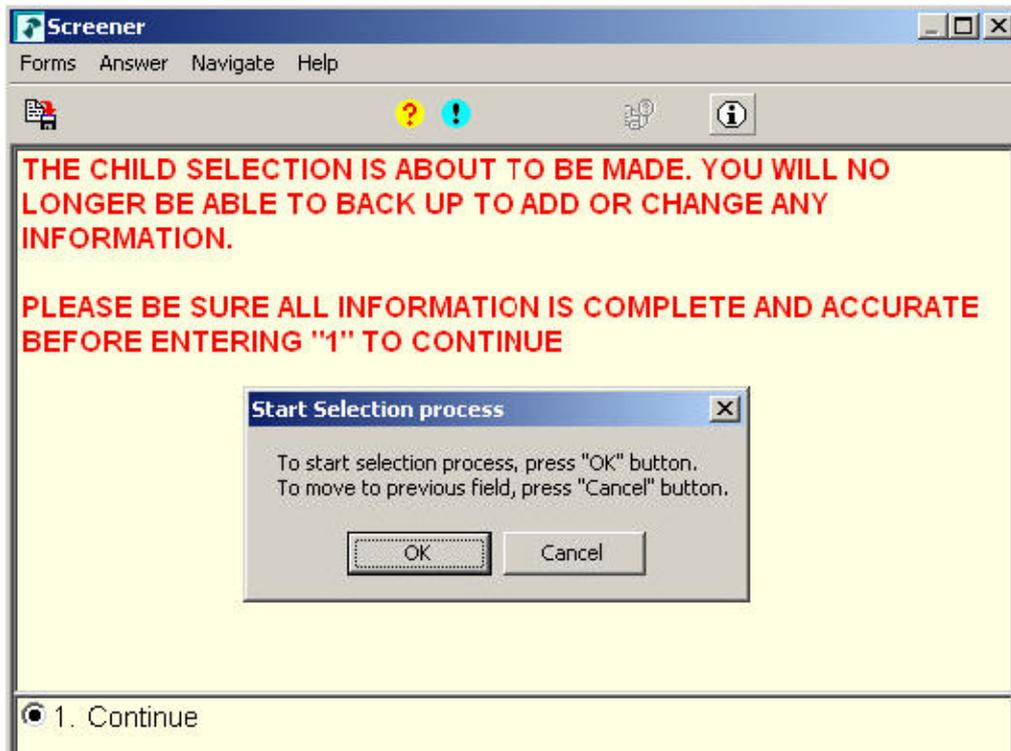


Figure 1. Call to Screener Selection Alien Router.

A Field Interviewer (FI) is asked to verify the child roster because the selection algorithms are executed only once to prevent changing the random number used to select a child, as shown on Figure 1.

After the FI clicks the OK button, the alien router reads ChildArray and adds records to three different arrays depending on a child relationship to a respondent. Records are sorted by age in each group and the selection algorithms are executed. As a last step in the alien router, information about selected children and sorted records is written to the Blaise database in SelectedChildren and OutChildArray, respectively. The FI receives a message that the selection is made and the Blaise interview continues.

3. Create Cases

3.1 Overview of Spawning Process

At RTI International a Case Management System (CMS) is used on FI laptops for many CAPI interviews. The CMS application allows Field Interviewers to update case status, enter comments, launch Blaise instruments, and synchronize the status of cases with a centralized SQL server database. A web-based Integrated Field Management System (IFMS) is used to assign and transfer cases between FI laptops and RTI. A web-based Control System (CS), on RTI's internal secure network, enables authorized staff to monitor the flow of data from the start of data collection through the creation of data files for analysis and delivery.

Sometimes, immediately upon completion of an interview, it is desired to launch a new, different interview for the same respondent. When this occurs, the new case is created on the FI laptop and is later automatically uploaded to a master Blaise database and IFMS during transmission.

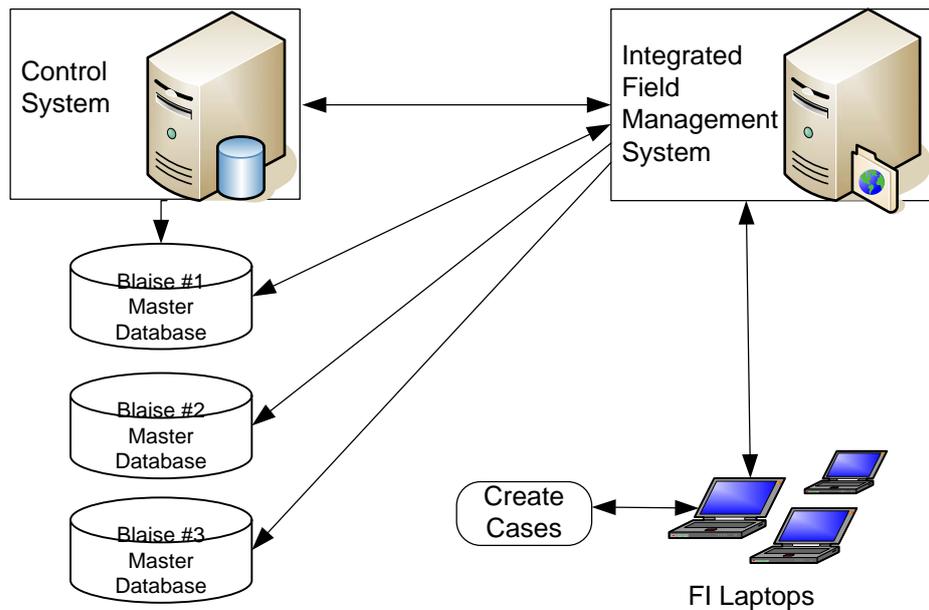


Figure 2. Data flow between FI laptop and RTI.

3.2 Create Cases Application

Typically, a Manipula setup is used by the CMS on FI laptops to spawn new cases upon the completion of an interview. However, in the described study Manipula was not well suited because of complicated requirements.

For cases where no children were selected, a new case should be created on the FI laptop to conduct an additional interview immediately upon completion of the screener interview. But for cases where one or two children were selected, cases should be created only in centralized Blaise databases to allow staff at RTI to prepare and send documents needed to conduct CAPI interviews with the respondent and children.

To meet requirements for the study, a Create Cases application was developed in Visual Basic using the BCP. It can run in one of three different modes (stored in a configuration file):

- **Production Mode on FI Laptop** – This spawns an interview immediately if desired and is used in the field.
- **Production Mode at RTI** – This spawns all CAPI interviews and produces special output files to be used by other systems (CS and IFMS) as shown in Figure 3. If cases were created, the application triggers an email to the RTI staff with information about the location of files that are ready for them to process in order to create documents for FIs.
- **Training Mode on FI Laptop** – This spawns all appropriate interviews that were supposed to be created after completion of the screening interview. It is used during training of FIs to show the whole process and allow training on the interviews that normally are not available on the laptop.

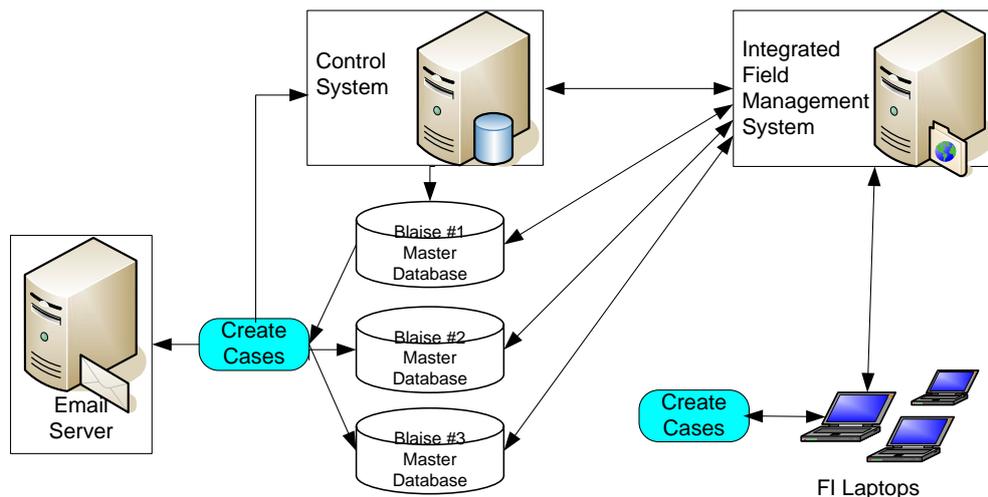


Figure 3. Data Flow between FI laptops and RTI with Create Cases application.

During the development stage of the study the data model of Blaise interviews changed many times. When a Manipula setup was used to create cases, a recompile was necessary for any data model changes. An advantage of using the BCP is that it allowed us to access information in one Blaise database and to use it to write data into another Blaise database independently of the data model versions involved in the exchange. Thus we made changes to the Create Cases application only when the fields needed to spawn new cases were changed, and that happened very rarely.

4. Fingertick Timer

During the CAPI interview in the described study, respondents are asked to participate in a blood sample collection. Three blood samples (“fingerticks”), collected 20 minutes apart, were required. The blood samples were to be collected concurrent with the ongoing interview. Instead of stopping the Blaise interview entirely during the waiting periods between samples, a method was devised to allow an independent Windows application to run timers to remind the FI to collect each blood sample, yet still allow the Blaise interview to proceed. This “Fingertick Timer” method involved a number of blocks defined in the Blaise instrument to facilitate usage of an alien router, a Fingertick Timer alien router, and a Fingertick Timer Windows application as shown in Figure 4.

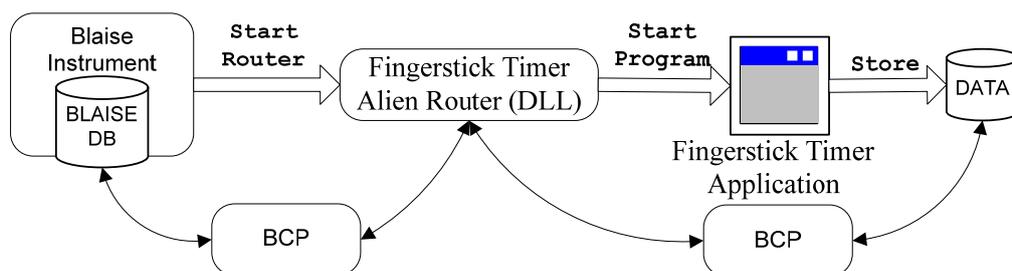


Figure 4. Fingertick Timer Invocation and Data Flow.

The fundamental tasks of the Blaise instrument are to:

- launch the Fingertick Timer alien router at some point during the interview, which in turn launches the Fingertick Timer Windows application
- at some point later in the interview, launch the Fingertick Timer alien router again to read the timer values collected by the Fingertick Timer Windows application
- store the timer values into the Blaise database

The fundamental tasks of the Fingertick Timer alien router are to:

- launch the Fingerstick Timer Windows application, passing the current Blaise case ID as a parameter
- read the comma-separated value (CSV) file output by the Fingerstick Timer Windows application and write the values into corresponding Blaise database fields
- log run-time information to a log file

The fundamental tasks of the Fingerstick Timer Windows application are to:

- display a pop-up window three times, 20 minutes apart, showing that it is time to collect a blood sample
- enable Breakoff of the blood sample collection (in case respondent refuses)
- record several date and time data points (exact date/time each 20 minute timer is started, exact date/time the pop-up window was presented, exact date/time each fingerstick was started, exact date/time each fingerstick was completed) and store these values in a separate CSV file for each respondent.

At some point during the Blaise interview, the respondent is asked to participate in blood sample collection, and if they consent the Blaise code calls the Fingerstick alien router, which in turn opens an instance of the Fingerstick Timer Windows application. The initial Fingerstick Timer application screen is shown in Figure 5.

The interviewer clicks on the “Begin Fingerstick” button when the fingerstick collection is started. The screen then changes to Figure 6. This screen counts up, in seconds, starting at zero. When the fingerstick collection is complete, the interviewer clicks on the “End Fingerstick” button and the Fingerstick Timer application starts a 20 minute timer then automatically minimizes itself so the Blaise interview is now showing on the screen. When the 20 minute timer expires, the Fingerstick Timer application pops up the window shown in Figure 5 again and the process repeats until three fingersticks have been collected. As each date/time value is captured, it is stored in a CSV file.



Figure 5. Initial screen



Figure 6. Time is counted up

At some later point in the Blaise interview, the Blaise code then calls the Fingerstick alien router again to get the results of the Fingerstick Timer application. First, the router ensures that the timer application is not still in progress, and if not it reads the CSV file and writes the values back into fields in the Blaise code.

Examples of .Net classes used to monitor calls to executables and log discovered problems or errors for the Fingerstick Timer implementation are provided in the Code Listing Appendix.

5. Bio-Tracking System (BTS)

5.1 Importing Field Data into BTS

When completed CAPI interviews are received at RTI, collected information is imported from the Blaise database during the nightly job at RTI into BTS by a .Net utility program, Load BTS. The program uses the BCP to open the Blaise database to access data collected in the Blaise interview and in the Fingerstick Timer application. This application reads more than one hundred fields per completed case and exports them into temporary tables in a SQL Server database as shown in the example below.

```
//Open Blaise Database
  BIAPI4A2.Database db = dbMgr.OpenDatabase(SourceDB);
  db.AccessMode = BIAccessMode.blamShared;
  db.Connected = true;
...
//Export Fingerstick Timer data
Time1 = db.get_Field("FngTmrResults.FngTmrResultsData[1].FSTimerEnd").Text;
Time2 = db.get_Field("FngTmrResults.FngTmrResultsData[2].FSTimerEnd").Text;
Time3 = db.get_Field("FngTmrResults.FngTmrResultsData[3].FSTimerEnd").Text;
...
//Export refusal answers
T_Sref1 = FormatData(db.get_Field("T_S.T_Sref[1]"));
T_Sref2 = FormatData(db.get_Field("T_S.T_Sref[2]"));
T_Sref3 = FormatData(db.get_Field("T_S.T_Sref[3]"));
...
```

The FormatData() function is used to export the value from the Blaise database and examine “Don’t know” and “Refusal” responses.

```
static public String FormatData(BIAPI4A2.Field obj)
{
  string p;
  if (obj.Status == BIFieldStatus.blfsDontKnow)
    p = "9";
  else if (obj.Status == BIFieldStatus.blfsRefusal)
    p = "8";
  else
    p = FilterNulls(obj.Value);
  return p.ToString();
}
```

After the completion of data importing steps, this program invokes an algorithm to identify the cases that are required to be loaded into BTS. The cases are separated based on sample type (blood spot or saliva). For blood spot, up to 3 samples on different color coded cards are collected with one or more fingersticks per card. For saliva a single sample is collected. A corresponding record is added to BTS for each collected sample.

5.2 Using Field Data in BTS

The application allows authorized staff at RTI to enter shipment details such as FedEx number, date shipped, time refrigerated, etc. Lab personnel who process blood spots and DNA look up the shipments by FedEx number or a special biospecimen ID number (BioID). The preloaded field data is used to verify information from FIs and simplify the process of entering additional information as shown in Figure 7.

Enter Shipment

Shipment Type:

Enter Bio ID:

(OR) Enter Case ID:

Date Shipped (mm/dd/yyyy):

FedEx:

Date Refrigerated (mm/dd/yyyy):

Time Refrigerated: (HH:MM AM/PM)

of Blood Spot Cards:

Master Status Code:

FI Name:

Preloaded data from field

Blood Spot

BIO ID	Card#	Sticks	Stick Time	Time Zone	Status	Date Received	Sex	Age
00766	Red	1	10:29:03	ET	B - Sample collected		F	26
00766	Blue	1	10:54:16	ET	B - Sample collected		F	26
00766	Yellow	1	11:19:36	ET	B - Sample collected		F	26

Finger stick times for each card, from the field

Color coded card description

Figure 7. Enter Shipment Screen

When laboratory personnel enter results on the form, preloaded data from the field such as fingerstick time can be compared against the real time that was shown on the sample. This provides additional verification, improves accuracy, and confirms that the labs are entering the receipts against the proper sample.

A number of BTS reports provide information about data collected from the field to monitor FI performance, case status, sample shipment and receipts, sample quality, etc. For example, the Cumulative Bio-Specimen Status report by FI shown below uses more than 60 fields passed from the completed CAPI interview to calculate the total number of completed eligible interviews, the total number of respondents who consented, those who refused, the total number of samples collected by case, and lastly those cases where blood spots were collected but have yet to be shipped to the lab. The report also shows quantity by case to indicate if the FI collected all three blood spots or one/two blood spots per case.

FI_Name	Overall - by Case										Quality - by Cards								Quantity - by Case			
	Completed eligible interviews		Total consented		Refused		Total Collected		Collected not shipped		Adequate		Marginal		Inadequate		Unusable		All 3 blood spots		1 or 2 blood spots	
	N	%	N	%	N	%	N	%	N	%	N	%	N	%	N	%	N	%	N	%	N	%
1. FI Name1																						
BLOOD SPOT	13	100.00	12	92.31	1	7.69	12	92.31	2	15.38	14	50.00	5	17.86	9	32.14	0		9	90.00	0	0.00
SALIVA	17	100.00	15	88.24	0	0.00	15	88.24	2	11.76	3	100.00	0		1		0	0.00	0	0.00	0	0.00
2. FI Name2																						
BLOOD SPOT	11	100.00	7	63.64	4	36.36	7	63.64	0	0.00	14	66.67	4	19.05	3	14.29	0		7	100.00	0	0.00
SALIVA	14	100.00	9	64.29	5	35.71	9	64.29	0	0.00	0		0		0		0		0	0.00	0	0.00
3. FI Name3																						
BLOOD SPOT	3	100.00	2	66.67	1	33.33	2	66.67	0	0.00	5	83.33	1	16.67	0	0.00	0		2	100.00	0	0.00
SALIVA	3	100.00	3	100.00	0	0.00	3	100.00	0	0.00	0		0		0		0		0	0.00	0	0.00
4. FI Name5																						
BLOOD SPOT	5	100.00	4	80.00	1	20.00	4	80.00	0	0.00	8	100.00	0	0.00	0	0.00	0		2	50.00	2	50.00
SALIVA	5	100.00	4	80.00	1	20.00	4	80.00	0	0.00	0		0		0		0		0	0.00	0	0.00
5. FI Name5																						
BLOOD SPOT	25	100.00	24	96.00	1	4.00	24	96.00	0	0.00	49	68.06	11	15.28	12	16.67	0		24	100.00	0	0.00
SALIVA	29	100.00	28	96.55	0	0.00	28	96.55	0	0.00	1	100.00	0		0		0	0.00	0	0.00	0	0.00

Figure 8. Cumulative Bio-Specimen Status report by FI

6. Generate Mandatory Report

Some studies at RTI require an alert be sent to staff so they will take appropriate action when certain interview answers of interest occur. This application is used to identify and report such events. Certain questions within several of the interview measures have been identified as indicators of concern. When particular responses are given to these questions in the CAPI interview, they trigger a series of follow-up questions. Once completed and

sent to RTI, the case is checked and run through the application. If the event is triggered, an email report is spawned and sent to the authorized staff alerting them to the need for further review.

Using the BCP, this Visual Basic application makes it possible to meet the requirement for checking 71 events and 538 variables, setting a flag in the Blaise database upon completion and sending an email report within 24 hours of receipt of the data.

Since there are many variables to be checked and to be reported, the application is written in a way that the code can be used with many variables. Events with the same structure but different question text are grouped in arrays.

```
'Define block names that have maltreatment flag
...
strFld(4) = "TCMC.TCM52"
...
strText(4) = "child ever physically abused"
```

When the application checks events, the base variable name of the event, FldName, is created from values defined in array strFld. The variable name of the follow-up questions is created by appending suffixes to the base name. The value from the Blaise database is assigned to a variable strTemp. The event descriptions are assigned to array strText.

```
'Check all elements in arrays
For i = 1 To 6
For j = 1 To 4
    'Define variable name of the event
    FldName = strFld(i) & "Arr[" & CStr(j) & "]"
    PrintName = "THL" & " - " & Mid(strFld(i), 6) & CStr(j)
    PrintName2 = "THL" & " - " & Mid(strFld(i), 6)
    If j = 1 Then
        FldName2 = strFld(i)
        PrintName2 = "THL" & " - " & Mid(strFld(i), 6)
        strLine = Space(4) & PrintName2 & " - " & _
            strText(i) & " - " & mBlaiseDB.Fields.Item(FldName2).Value & " (" & _
                mBlaiseDB.Fields.Item(FldName2).Text & ")"
        Print #intFile, strLine
    End If
    'Get value of the follow-up questions
    If mBlaiseDB.Fields.Item(FldName & "a").Status = blfsDontKnow Then
        strTemp = cDK
    ElseIf mBlaiseDB.Fields.Item(FldName & "a").Status = blfsRefusal Then
        strTemp = cRF
    ElseIf mBlaiseDB.Fields.Item(FldName & "a").Value > 0 Then
        strTemp = mBlaiseDB.Fields.Item(FldName & "a").Text
    End If
    If Len(strTemp) > 0 Then
        strLine = Space(4) & PrintName & "a" & " - " & _
            "age when this happened - " & strTemp
        Print #intFile, strLine
    End If
    ...
Next
Nex
```

Below is the example of a mandatory report. The report displays an abbreviation of a section's name (THL), a variable name (TCM52, TCM521a, etc.), a variable description and the value of the answer.

```
THL - TCM52 - child ever physically abused - 1 (Yes)
THL - TCM521a - age when this happened - 11
```

THL - TCM521b1 - perpetrator - acquaintance - female - adult
THL - TCM521g - number times this person did this - 2

The configuration file is a file containing initial settings for an application. Using a configuration file helps make testing more efficient. There are options that specify whether a flag would be set, whom email should be sent to and the location of the input Blaise database. Transition from testing mode to production mode goes effortlessly with this configuration file.

7. Overnight Process

To accomplish successful exchange of the data between the different systems and the programs described earlier, an automated nightly job was set up. It was essential to execute a number of different programs in a predefined sequence to have IFMS, CS, and BTS systems updated correctly.

As a first step, cases from the field are loaded into Blaise master databases. Then, the three described programs, Create Cases, Load BTS, and Mandatory Report, are called sequentially to process the data as shown in Figure 9.

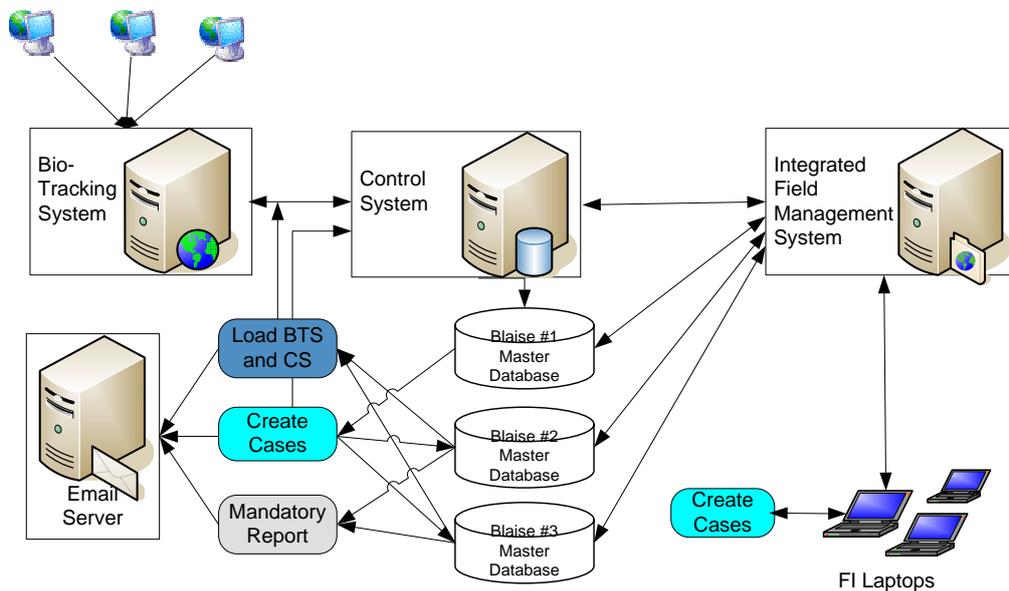


Figure 9. Data Flow between FI laptops, RTI, and Labs.

If necessary, the programs send emails to appropriate staff to inform them that a required action should be taken. Lastly, the newly created cases are added to IFMS for transmitting to FI laptops.

8. Conclusion

For the described study, the following advantages of the BCP were used:

- Ease implementation of sorting and randomization
- Fast programming of complicated tasks
- Ease of debugging the applications
- Effortless learning for experienced Visual Basic and .Net programmers
- Ease of adding already tested common classes and functions

The BCP reduces programming time for creating processes to pass data from Blaise databases into SQL Server databases, Excel, and other types of output files. For complex CAPI surveys that involve interviews of selected household members, collection of biospecimens, psychological tests, and special reports, usage of the BCP helps make data collection efficient and accurate. The BCP is being used at RTI extensively to replace Manipula when possible.

9. Acknowledgements

The authors would like to acknowledge Joe Nofziger and R. Suresh of the Research Computing Division at RTI International for help with this paper and for input into how to address some of the requirements requested.

10. References

Lilia Filippenko, Joseph Nofziger, Mai Nguyen and Roger Osborn (2006): Methods of Integrating External Software into Blaise Surveys, Proceedings of the 10th International Blaise Users Conference, Arnhem, Netherlands, May 2006.

M. Rita Thissen, Sridevi Sattaluri and Lilia Filippenko, (2007): Using the Blaise® Alien Router for Audio-Recording of In-Person Interviews, Proceedings of the 11th International Blaise Users Conference, Annapolis, USA, September 2007,

Appendix: Code Listings

Code Listing 1 – Log File Class

```
class clsLog
{
    private String LogPath;
    public clsLog(String BlaiseDBPath, String strClassName)
    {
        LogPath = BlaiseDBPath + @"\Log";
        DirectoryInfo di = new DirectoryInfo(LogPath);
        if (!di.Exists) di.Create();
        LogPath += @"\";
        LogPath += strClassName;
        LogPath += @"_Log.txt";
    }
    public void Log(string message)
    {
        StreamWriter writer = new StreamWriter(LogPath, true);
        writer.WriteLine(DateTime.Now.ToString() + ": " + message);
        writer.Flush();
        writer.Close();
    }
    public void LogException(Exception ex)
    {
        string strException = "EXCEPTION:" + Environment.NewLine + ex.Message;
        strException += Environment.NewLine + "STACK TRACE:" + Environment.NewLine + ex.StackTrace;
        if (ex.InnerException != null)
        {
            strException += Environment.NewLine + "INNER EXCEPTION:" + Environment.NewLine +
ex.InnerException.Message;
        }
        this.Log(strException);
        System.Windows.Forms.MessageBox.Show(strException);
    }
}
```

Code Listing 2 – Alien Router Class Log File Usage

```
public class FingerstickTimerRouter
{
    ...
    private string BlaiseDBPath;
    private clsLog objLog;
```

```
public void Run(BIAPI4A2.Database data, BIAPI4A2.DepState state)
{
    if (state.AlienRouterStatus != BIAlienRouterStatus.blrsPreEdit) return;
    this._data = data;
    this._state = state;
    BlaiseDBPath = data.DataFileName.Substring(0, data.DataFileName.LastIndexOf(@"\"));

    objLog = new clsLog(BlaiseDBPath, this.ToString().Substring(this.ToString().LastIndexOf(".") + 1));
    objLog.Log(_data.KeyValue + ": " + String.Format("Instance started on {0} {1}",
DateTime.Now.ToShortDateString(), DateTime.Now.ToShortTimeString()));

    try
    {
        ...

    }
    catch (Exception ex)
    {
        objLog.LogException(ex);
    }
    finally
    {
        objLog.Log(_data.KeyValue + ": " + "Instance ended");
    }
}
}
```

BLAISE in Macedonia-Census of agriculture 2007¹

Liljana Taseva, Mira Deleva, State Statistical Office of the Republic of Macedonia

This paper is dedicated to the expert from Slovenian Statistical Office, M.Sc Pavle Kozjek, to whom we are deeply grateful for the mutual collaboration.

1.Introduction

Census of agriculture was one of the main actions conducted in State Statistical Office of Republic of Macedonia (SSO) in 2007. It was a process of collecting, processing, assessing, analyzing and disseminating of data related to agricultural situation in Republic of Macedonia. In the period after the Second World War, on the territory of Republic of Macedonia the Census of Agriculture was conducted twice, the first time in 1690 and the second in 1969. Data collected through the census are of great importance for creating strategies for the future national and local development. Census of agriculture in 2007, on the territory of Republic of Macedonia was conducted in the period from 01-15.June.2007.

SSO prepared following census instruments:

- Survey form for individual farmers, as well as form for farms registered as business subjects;
- Control form for registering the census units;
- Form for post census survey;
- Methodological guides.

Data collection was conducted in traditional way, by interviewers on the field and using paper forms.

According to planed activities and deadlines, applications for data processing (data entry, control, aggregation, derivation of some variables, tabulation) were created. Considering long experience in using Blaise at SSO, it was decided to use it for entering, checking and editing of data collected via Census of agriculture in 2007.

Presently, the main tool used for data entry at SSO is Blaise .The first application has been developed in 1999 when this software was used only for opinion poll based surveys, but latter SSO has started using Blaise for the rest of surveys as well, including Census of agriculture. Experience from using the Blaise increased with its continuous use, especially for creating of data entry forms as well as for establishing of some uniformed procedures.

2. Brief overview of data entering and processing

The Census of agriculture data entry began one month after the end of the fieldwork, i.e. after publishing the first results.

Before data processing took place in SSO, some preparatory work was to be done, such as: enhancing the IT environment; developing an applicative software for data processing; providing working space for data processing; choosing staff for visual control, coding, entering and correction of the paper material; choosing staff for surveillance of the working process; providing staff training; preparing manuals for data processing and applicative software.

According to the plan of activities and deadlines, SSO was equipped with server and 46 working stations dedicated for this activity. Workstations were placed in the premises provided for data processing. Local area network in Windows environment and software tool Blaise 4, were used for completing the process of entering

¹ Views expressed in the paper do not necessarily reflect the views of the State Statistical Office

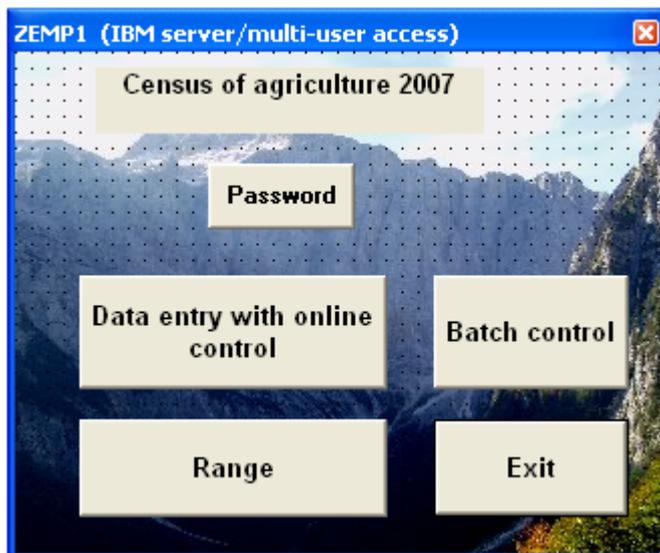
data of Census of Agriculture. Records affected during the process of data entry counted 197500 farms, and for each and every one of them were entered approximately 980 variables.

Applications for data entry, data control and data editing, developed in Blaise, were placed in special server folder (**POPZEM07**). Access to data and applications on server was controlled via user interface, developed in VB, by typing account name and password.

Several applications were developed in Blaise 4 :

- entrance of identification data for the carrier of the farm;
- entrance of data for the farm related with online control;
- Blaise data models for meta data files, such as: municipalities, settlements, census districts, streets, classification of activities (NACE REV. 1);
- manipula for transfer of meta data files from .txt to .bdb format;
- manipula for batch control;
- manipula for coping of the program for data entry along with data, from one model to another;
- manipula for coping Blaise file to ASCII files;
- manipula for data editing in DB2 database;
- manipula for daily reporting for data entry on the data entry person level and summary report for data entry on the data entry person level for given period of time as well as total summary reports.

Figure 1: user interface, developed in VB6, used for accessing applications in Blaise for data entry and control.



2.1 Application for data entry

User interface for data entry was designed according to the questionnaire design. It was accomplished by standardized code of blocks and tables, fields were textual and numerical.

Every farm had its own identification, recognized as primary key, which was a complex key composed of municipality, census district and ordinal number of the farm. There was an online control of the existence of such municipality and census district regarding meta data files. In the case of incorrect choice of municipality or census district, an error dialog appeared.

Application design was made to follow the flow of the questions. Data were entered in the questionnaire order, from question to question, from table to table. In the table where data of household members were placed, all jumps were programmed, so data entry person had to follow only filled fields of the questionnaire. Positioning on the proper field was automatically guided.

The same application was used for data entry and for data correction. Hence, in the application were implemented all logical-mathematical controls. Controls were very complex, so part of them were implemented

during the data entry process, and the other part after the data was entered. Part of the logical-mathematical controls were implemented as on-line control, so the entering of illogical data was reduced. In the case where illogical data were entered, error dialog appeared with error specification. Automatic positioning on the error fields was done and data entry person had to correct the value of positioned fields.

2.1.1.Example of on-line control in the table for household members

RULES

```
IF lice[I].cl_5k=child THEN
    (lice[i].cl_4k < lice[1].cl_4k) AND (((lice[1].cl_4k - lice[i].cl_4k) < 50) and
        ((lice[1].cl_4k - lice[i].cl_4k) > 14))
    "child ^lice[i].cl_4k difference with carrier
has to be less than 50 years and greater than 15 years ^lice[1].cl_4k      Error 106_tab40
ENDIF
```

2.1.2 Example of on-line control on table 4 – land in household disposal

AUXFIELDS

c4_1: 0..99999999

c4_2: 0..99999999

.

FIELDS

greska : ARRAY [1..100] of STRING[1]

.

RULES

```
c4_1:=((tabela4.P4[1].k4_h*10000+ tabela4.P4[1].k4_m) +
    (tabela4.P4[2].k4_h*10000+ tabela4.P4[2].k4_m) -
    (tabela4.P4[3].k4_h*10000+ tabela4.P4[3].k4_m))
c4_2:= tabela4.P4[4].k4_h*10000+ tabela4.P4[4].k4_m
```

```
IF (c4_1)=(c4_2) THEN
    greska[1]:='1'
ELSE
    error involving (tabela4.P4[4].k4_m) "Error 1 – sum error in table4"
ENDIF
```

In this case in the field for certain error, the correct combination is signed as in the case above with greska[1]:='1' and for the incorrect combination appears error dialog for the combination that has to be corrected at the moment of data entrance (on-line).

The rest of the controls were executed during the data entrance by marking incorrect and correct combinations in the following way:

RULES

.

.

```
IF tabela16.P16[19].k16_h*10000 + tabela16.P16[19].k16_m =
    Tabela17.P17[14].k17_1h*10000 + Tabela17.P17[14].k17_1m THEN
    greska[11]:='1'
ELSE
    greska[11]:='0'
ENDIF
```

Once data entry for each municipality was accomplished, an application (manipula) for batch control was executed, in order to create a list of erroneous records. Each erroneous record consisted of identification data (primary key) and type of the error. Since data entering created one .bdb file, in application (manipula) for batch-control existed option for its executing for one or more chosen municipalities. Data control and correction were executed until database was cleaned of errors.

That is described in the code below:

```

Settings
DESCRIPTION = 'List of errors - zemp1'

USES
    InputMeta 'zemp1'

DATAMODEL outf
    FIELDS
        oneline:STRING[100]
ENDMODEL

INPUTFILE InFile: InputMeta ('zemp1',BLAISE)
OUTPUTFILE OutFile: outf ('zemp1_greski.txt', ASCII)
SETTINGS
MAKENEWFILE=yes

AUXFIELDS(global)
i:INTEGER
k:INTEGER

MANIPULATE
IF recordnumber (infile)=1 THEN
oneline:='-----' outfile.WRITE

oneline:='ZEMP-1 Pogresni podatoci na den:'+ datetostr(sysdate)+' vo: '+timetostr(system)
outfile. WRITE
oneline:='
outfile. WRITE
oneline:='opstina popisen krug stopantvo greska ' outfile. WRITE
    oneline:='
        outfile. WRITE
ENDIF

IF (ops = '0035') or (ops = '0116') or (ops = '1414') or (ops = '1228')
or (ops = '1643') or (ops = '1872') THEN

    FOR i:= 1 to 12 DO
        IF greska[i]<'1' THEN k:=i
oneline:= ops + ' ' + krug + ' ' + stop + ' ' + Greska ' + FORMAT(str(k),2,right)
        OutFile.WRITE
        ENDIF
    ENDDO

    FOR i:= 14 to 17 DO
        IF greska[i]<'1' then k:=i
oneline:= ops + ' ' + krug + ' ' + stop + ' ' + Greska ' + FORMAT(str(k),2,right)
        OutFile.WRITE
        ENDIF
    ENDO
ENDIF
ENDIF

```

Example of created report from batch control

municipality	settlement	Census district	farm	Error
1040	BEROVO	059	013	Error 2
1040	BEROVO	059	034	Error 88
1040	BEROVO	059	034	Error 89
1040	BEROVO	060	004	Error 27
1066	BITOLA	060	012	Error 66

1066	BITOLA	075	002	Error 12
1066	BITOLA	068	004	Error 6
1678	KUMANOVO	261	021	Error 5
1678	KUMANOVO	261	021	Error 45

Identification data regarding the farm holder such as: name, surname, personal ID, municipality, settlement and address (street and house number), phone, were placed in separate file apart from the rest of data regarding the farm, hence the necessity for checking the range of the first file against the range of the second file and vice versa. This control was executed continuously during the data entry.

3. Following carried out procedures

Backup of the .bdb file of data-track, was created at the end of each working day. Also, the contents of the folder POPZEM07, where applications and data resided,

were copied into another server folder named ARHIVA, on the daily basis. It was done in order to provide data entry to flow continuously and simultaneously with the rest of data processing procedures, such as batch-control of data, creating necessary reports for the process of data entry etc. This was of the great importance for the batch control because in the folder named ARHIVA resided data of data entry from the previous day, enabling the possibility for data entry and batch control to run simultaneously and control of data to end up at the same time with data entry.

After data entry and control were accomplished, followed the process of migrating data to DB2 database. Because of the lack of the modules for direct communication between Blaise files and DB2, manipula for transferring data from Blaise file to ASCII file was developed. With this manipula data entry file is transformed into several .txt files, mutually connected by primary key, where data were separated by delimiters (>).

Basic structure of this application is given in the example below:

```

SETTINGS
DESCRIPTION = 'BLAISE to ASCII'

USES
InputMeta 'zemp1'

DATAMODEL zemp1_d1
  FIELDS
    OPS "Opstina": STRING[4]
    KRUG "Popisen krug": STRING[3]
    STOP "Stopanstvo": STRING[3]
    TABELAID :BID
    tabela4 :tab4
  .
  Tabela18: tab18
  .
  .
ENDMODEL

DATAMODEL zemp1_d2
  FIELDS
    OPS "Opstina": STRING[4]
    KRUG "Popisen krug": STRING[3]
    STOP "Stopanstvo": STRING[3]
    TABELAID :BID
    Tabela19: tab19
  .
  Tabela21: B21

```

```
.  
ENDMODEL
```

```
DATAMODEL zemp1_d3
```

```
  FIELDS  
    OPS "Opstina": STRING[4]  
      KRUG "Popisen krug": STRING[3]  
      STOP "Stopanstvo": STRING[3]  
      TABELAID :BID  
      Tabela22: tab22  
    .  
    .  
      Tabela32: B32
```

```
.  
ENDMODEL
```

```
DATAMODEL zemp1_d4
```

```
  FIELDS  
    OPS "Opstina": STRING[4]  
      KRUG "Popisen krug": STRING[3]  
      STOP "Stopanstvo": STRING[3]  
      TABELAID :BID  
      Tabela33: tab33  
    .  
    .  
      Tabela46: tab46
```

```
.  
ENDMODEL
```

```
INPUTFILE InputFile1:InputMeta ('zemp1', BLAISE)
```

```
OUTPUTFILE OutputFile1: zemp1_d1 ('zemp1_file1.txt', ASCII)
```

```
SETTINGS
```

```
SEPARATOR = '>'
```

```
OUTPUTFILE OutputFile2: zemp1_d2 ('zemp1_file2.txt', ASCII)
```

```
SETTINGS
```

```
SEPARATOR = '>'
```

```
OUTPUTFILE OutputFile3: zemp1_d3 ('zemp1_file3.txt', ASCII)
```

```
SETTINGS
```

```
SEPARATOR = '>'
```

```
OUTPUTFILE OutputFile4: zemp1_d4 ('zemp1_file4.txt', ASCII)
```

```
MANIPULATE
```

```
  OutputFile1.WRITE
```

```
  OutputFile2.WRITE
```

```
  OutputFile3.WRITE
```

```
  OutputFile4.WRITE
```

Scripts for creating tables in DB2 database and import of .txt files into DB2, were made by using CAMELEON. From DB2, data were exported into SAS, which was used as a tabulation tool. During the preparation of data for tabulation, additional variables were derived in order to easy the process of tabulation in technical terms and to improve efficiency in finalizing output results. Applications for aggregation of data and tabulation of the output results were developed. Afterwards results were analyzed and published.

For the first time it was achieved that conducting, processing and publishing of the Census data was be accomplished within the same calendar year.

WEB dissemination of Census of agriculture 2007 data was realized through software tool PC AXIS in which was offered a greater part of published data.

4. Post Census survey – data entry

In order to check and confirm the range and quality of data collected from Census of agriculture in 2007, statistical survey based on sample of 50 census districts over the territory of Republic of Macedonia was conducted independently, right after ending the field work from 16-20/06/2007 and included 2% of total farms. The survey was conducted in traditional way by repeated visits and collection of data. In Blaise was developed application, with all necessary on-line data controls implemented. Copy of application was created with different name of .bdb file, in order to have the same application twice, so the double data entrance could be performed. Manipula was developed for comparing data from the two files. After two files were matched, data from post Census survey were processed in SAS.

5. Conclusion

Conducting of the Census of agriculture in 2007 was successfully accomplished task and for the first time it was achieved that conducting, processing and publishing of data of Census to be carried out within the same calendar year. Great contribution to this task gave Blaise 4.7, as a software tool in which were developed applicative solutions that provided overcoming of some deficiencies in the previous versions of Blaise (for instance hospitalizing of data etc)

Because of the successful use of this software, SSO is tending to extend the use of Blaise from opinion poll based surveys to the rest of statistical surveys. Experience of using Blaise for Census of agriculture confirms that this software can be used as a tool for data processing for other mass actions.

Quality assurance through Computer Audio- Recorded Interviewing (CARI): The Statistics New Zealand Case Study

Chris Seymour, Statistics New Zealand

Background

In 2006 the Data Collection Standards and Consultancy (DCSC) conducted a research project to identify and establish a set of generic interviewer best practice behaviours for Statistics New Zealand.

The research project recognised that while a number of quality assurance mechanisms were currently utilised within the Statistics New Zealand field collections business unit, they were employed on an ad hoc rather than systematic approach, which was not easily reported on or demonstrated to external parties.

These practices consisted of:

- An annual field check conducted by the interviewer's regional field supervisor.
- Checking of completed interviews on receipt in the office.
- Regular field and completed survey checks of newly recruited interviewers.
- Rotation of interviewers during quarterly or longitudinal surveys so that more than one interviewer collected data from the respondent and anomalies in collection practices could be identified.
- Informal 'teaming' of inexperienced interviewers with more experienced interviewers.
- Training.

The project set about compiling a set of standards brought together from a variety of sources including market research organisations, international statistical agencies and social science research. These standards were then presented to and endorsed by the Statistics New Zealand Standards Governance Board with a view that the implementation of these standards would:

- Increase confidence in data quality, for both stakeholders and the public.
- Increase confidence that official statistics are collected via sound methodology.
- Increase consistency across interviewers, minimising the risk of interviewer related error and falsification of data.
- Increase confidence in the professionalism of our field force.
- Increase respondent cooperation and compliance.

The Best Practice Standards (BPS) for Interviewers initiative, which was part of the wider Best Practice Field Collections project, adopted the DCSC's standards as a foundation from which to develop a revised set of field interviewer best practice standards and practical quality assurance mechanisms, to address the issues raised by the research project.

The BPS initiative identified twelve recommendations to quality assure interviewing standards, of which the use of Computer Audio-Recorded Interviewing (CARI) was one. CARI provides the unobtrusive recording of verbal communication between an interviewer and respondent as the survey is conducted.

Computer Audio-Recorded Interviewing (CARI)

Traditional methods used to ensure the quality of field collected data are often expensive, logistically challenging and difficult to qualify.

Traditional quality assurance method	Problem
Interviewer Rotation	Logistically challenging and expensive for remote areas.
Interviewer Observation	Expensive and time consuming
Post survey respondent phoning	Difficult to qualify results as often the respondent doesn't remember answers

Traditional quality assurance method	Problem
Comparison of survey audit information	Subjective views are reached

The use of a CARI solution aids in meeting the following quality assurance objectives for field collected survey data:

- Quality assurance of the data collected by computer assisted personal interviewing, through
 - Validation of the authenticity of the interview
 - Capture of the full response for a question or set of questions
- Monitoring of adherence to standards by interviewers, through
 - Monitoring of compliance to best practice standards.
 - Providing information to help identify areas for interviewer development.
- Evaluation / confirmation of the effectiveness of questionnaire items in the field
 - Ability of interviewers to read the survey questions in a fluent and understandable manner.
 - Questions elicit the desired response from the respondent (without the need for explanation or probing by the interviewer).

The approach allows for the survey supervisor or administrator to selectively target new questionnaires or questionnaire sections without adversely affecting the information collected as the survey is conducted. The audio information collected through CARI provides evidence that an interview is conducted correctly and to organisational standards, while building up a raft of material to aid in formal interviewer training, which in turn increases the quality of the interviews conducted.

Statistics NZ first became aware of CARI as a quality assurance tool from a demonstration by Rita Thissen of Research Triangle Institute (RTI) at the 2007 International Field Director's and technologies Conference.

Help and guidance from RTI after the conference resulted in the development of a CARI prototype at Statistics New Zealand and has since been further developed into an end to end system.

Implementing CARI at Statistics New Zealand

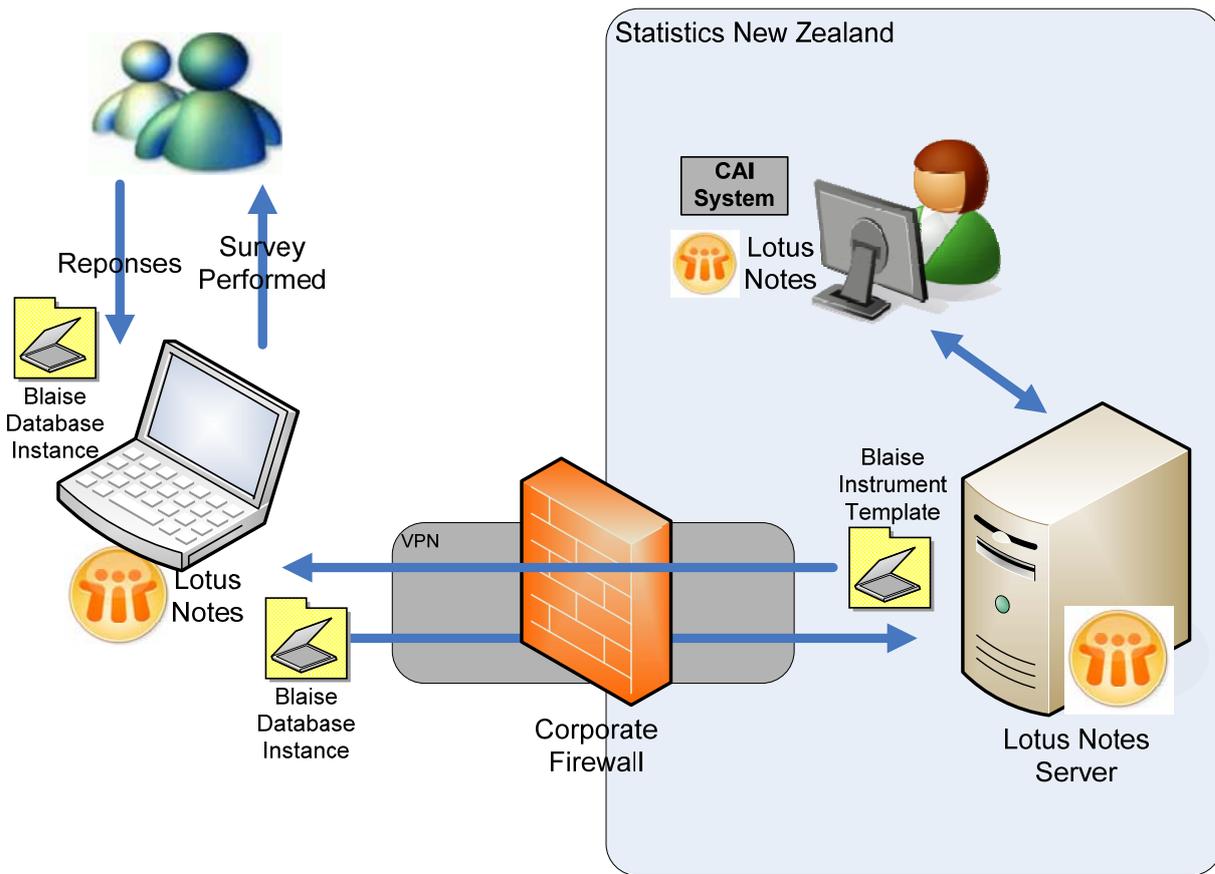
Statistics New Zealand has invested heavily in making Blaise its Computer Assisted Personal Interview (CAPI) tool. This meant that any work done to implement a CARI based solution into our interviewing suite would need to seamlessly integrate with Blaise and other survey management systems.

The current Computer Assisted Interviewing (CAI) system utilises Lotus Notes as a portal that allows collection supervisors to assign cases to the interviewer who will perform the interview.

The interviewers use wireless broadband to download the case list assigned to them and with it the Blaise instrument and associated files to use for that survey.

When the interview begins a instance of the Blaise database is created for the respondent being interviewed and completed as the survey progresses.

Once the interviewer has performed a number of interviews they will replicate with the office system and upload the completed Blaise database into the CIA system. At this time changes to the case list assigned to the collection agent will also be downloaded.



Solution Design

The first step towards a CARI implementation that integrates into Blaise was to decide the Blaise integration strategy. Two choices are available:

1. Component Object Model (COM)

COM is an interface standard created by Microsoft that allows libraries of functionality to be implemented. Using COM allows functionality that is not currently supported in the native Blaise product to be implemented through two extension points available in the Data Entry Program (DEP). These extension points are referred to as:

- Alien Routers – Executed when a question is given the focus within the DEP and that question is tagged with an Alien Router statement.
- Alien Procedures – Executed by the RULES section of the data model within the DEP.

2. Extension of the Audit Trail

Blaise provides an audit facility that records the field values and movements within the questionnaire as the survey is conducted.

The source code, written in the development language Delphi, and example audit trail functionality are provided free with the Blaise system.

To help understand these integration options two short implementation prototypes were started, one extending the DEP through the use of the Alien Router keyword, and the second extending the audit trail example provided with the Blaise system.

After careful consideration of the prototype solutions it was decided to continue with the approach that extended the Audit trail functionality. A number of factors lead to this decision:

1. COM based technical issues were experienced resulting in instability of the prototype.
2. Use of the Alien Router approach required that the Blaise instrument be modified to attach the ALIENROUTER keyword to the blocks that will execute the external functionality. The Audit Trail approach required no modification of the instrument.
3. The Audit Trail approach was familiar to our internal Blaise developers as it had been used previously to provide multilingual support for questionnaires.
4. The base audit trail functionality had been stable for a number of years.

Design elements:

Before any development was done it was identified that the audit trail example, which is the base for our CARI implementation, used a procedural based development style which is arguably more difficult to read and therefore understand, and was a practice that as an organisation were moving away from.

To ensure that this application met our internal development practices and standards it was decided that the code base should be re-written using Object Oriented Programming (OOP) techniques, making the functionality easier to manage, maintain and extend. The remade audit trail code forms the base of the SNZAudit trial, retaining all its original functionality.

One of the known issues in relation to implementing an audio recording feature was the management of the audio files once generated. This posed problems not only for the storage of the audio files on the laptop, but the time and cost required transferring them back into the organisation.

Compressing the audio files was seen as the only option; however this raised a significant technical challenge. After some research a third party product called Active Audio Record (AAC) was found which allowed the audio stream to be recorded using one of a number of formats, including compressed formats like wma and mp3.

In order to identify the audio file generated for a particular question a robust naming convention was created using identifiers from the Blaise instrument and CAI system which together form a unique string that is used as the file name. The format is as follows:

```
<SurveyID><CaseID>__<SurveyType><SurveyVersion>_<CaseID><QuestionnaireID>_<Module>.<Series><index>.<Question><index>.WMA
```

Where:

Format Part	Description	Source
SurveyID	The survey identifier	CAI
CaseID	The unique case identifier	CAI
SurveyType	The survey type, (H)ousehold or (P)ersonal	CAI
SurveyVersion	The survey version	CAI
QuestionnaireID	The Questionnaire identifier	CAI
Module	The name of the blaise module	Blaise
Series	Series is the name of the blaise series or table	Blaise
Question	The name of the blaise question	Blaise
Index	An optional number identifying a repeating or sliding question	Blaise

Solution Usage

An important aspect of the Blaise integration approach that was adopted is that CARI does not affect the design or development of the actual Blaise instrument. This allows CARI to be used with any new or existing Blaise instrument by simply configuring the Blaise instrument to use our CARI audit trail functionality.

In order to use an Audit Trail routine Blaise must first be configured with the Audit Trail dll to use. To do this the Mode Library Editor is used (from the Control Centre menu -> Tools -> Modelib Editor).



From this dialog ensure the 'Make audit trail' checkbox is selected and specify the SNZAudit.dll as the dll to use.

The SNZAudit trail system has extended the Audit Trail Information file (.AIF) file to provided configuration items at runtime for both the audit and audio recording functionality. Upon execution of the Blaise instrument the system tries to configure itself from a file located in the same directory and with the same name as the blaise data model (.BMI) file. If this file can not be found then a second configuration attempt is made via a file called AuditTrail.AIF. If this file can not be found then the system is configured using default values.

Solution Configuration

With the CARI functionality built into the audit trial system additional configuration items are available as explained below.

The original options required by the Audit Trail functionality remain unchanged:

Audit Trail	Default	Comment
Enabled	True	Turns Audit trail on / off
Leavefield	True	Records details on leaving a field
Enterfield	True	Records details on entering a field
Action	True	Records details on an action (actions are menu options and control keys)
Keystroke	False	Records every single keystroke
Classify	False	
Lookup	False	
ErrorDlg	False	Records all error's that occur
FilePerForm	False	Creates a audit file for each 'form' (A file gets created called <Primary key> + .adt
Extra	False	Writes 'extra' info like username, metafile, etc.
FlushFile	False	Flushfile means the file is written straight away rather than waiting for the network to write from the buffer...
Mouse	False	Track mouse cursor
PreciseTiming	False	Shows 1000 th of a second
KeyTranslation	False	Handles double byte character sets
WriteLean	False	Writes audit information in an abbreviated format
KeyStrokeOneLine	False	Writes keystrokes on only one line in audit file
AuditFolder	Data dir	Folder in which to save audit files
AuditTempFolder	Data dir	Folder to store temporary audit files

Additional configuration items required by Statistics New Zealand's implementation of CARI:

CARI	Default	Comment
Enabled	False	Turns the recording functions on / off
WavAsKey	True	Adds the Key to the filename
Device	First found	Name of the recording Hardware device
Format	First found	MP3, WMA, OGG, VOX, AU, AIFF, MP4, FLAC
WaveFormat	First found	e.g. "5kBPS, 8kHz, Mono"
Mixer	First Found	e.g. "Front Mic"
WriteToAudit	True	Write's into the audit trail recording details.
SeperateFiles	True	Stores data in separate audio files
WavFolder		Folder the audio files are to be written to, if no value is here will default to Audit folder.
StartQs		List of 'start questions' separated by ';' NOTE: This is the FULL question name
EndQs		List of 'end questions' separated by ';'
MaxTime	60	Max time in seconds for each question.
ScriptStart		The start question of any CARI script
ScriptFile		A 'rich text' file that contains the content text to display.
ScriptConfirmText		The confirmation answer text to display
ScriptCancelText		The cancel recording answer text to display
MaxFiles		Maximum number of recordings per case
Random		Random chance for recording (decimal .5 = 50%)

Optional configuration item used to assist feature development:

Debug	Default	Comment
Enabled	0	Turns on debugging mode. When enabled a dialog window sits over the Blaise instrument displaying the audit trail and other Blaise information in real-time.

Recording Consent

Before a survey that has CARI enabled is started consent for the interview to be recorded is required from the respondent. To capture consent the interviewer is presented with a Blaise-like screen which presents the content question from the Rich Text Format (.RTF) file configured in the audit information file.

Blaise 4.8 Data Entry - C:\Users\Administrator\Desktop\Test\test

Forms Answer Navigate Options Help Stop Recording Debug On

For quality assurance and training purposes, short parts of this interview will be recorded and listened to by supervisory and quality assurance staff, and questionnaire developers.

Do you give your consent for the recording to occur?

Yes
 No

CARI 1

Once consent has been granted by the respondent the survey will proceed as normal. If consent is denied or withdrawn at some later point after the survey has begun then a reason for this action is captured through a popup window and stored in a separate file which is associated with this interview.

Recording is manually stopped through an additional menu item attached into the Blaise menu structure.

The action of stopping the recording process causes any audio files recorded up to that point to be removed.

Recording Questions

A number of options are available to determine exactly how and which questions are recorded when CARI is active.

The list of questions available to be audio recorded is configured through the CARI .AIF file using the 'StartQs' and 'EndQs' configuration items. These items hold a semi-colon separated list of the full Blaise question identifier.

Generally these two lists are the same and cause a recording to be started when the question is entered and stop when the question is left. By specifying different start and end questions within these configuration items enables a range of questions to be recorded. Each start and end set produces a separate audio file.

To provide a random element to which question is recorded a value between 0.05 and 0.50 can be assigned to the 'Random' configuration item. This value determines the likelihood that the question will be recorded. A default value of zero will ensure all the configured questions are recorded.

The length of a given recording can also be determined through the .AIF file. The 'MaxTime' configuration item sets the maximum durations, in seconds, of any individual recording.

The configuration item 'EndRecQ' specifies the Blaise question that informs the CARI system that no more recording is required. When this question is complete a file is created that stores an audio recording complete status. If the interviewer traverses back through the instrument, or the interview is stopped and restarted, audio recording will not be restarted as a check is made to see if the audio recording complete file exists.

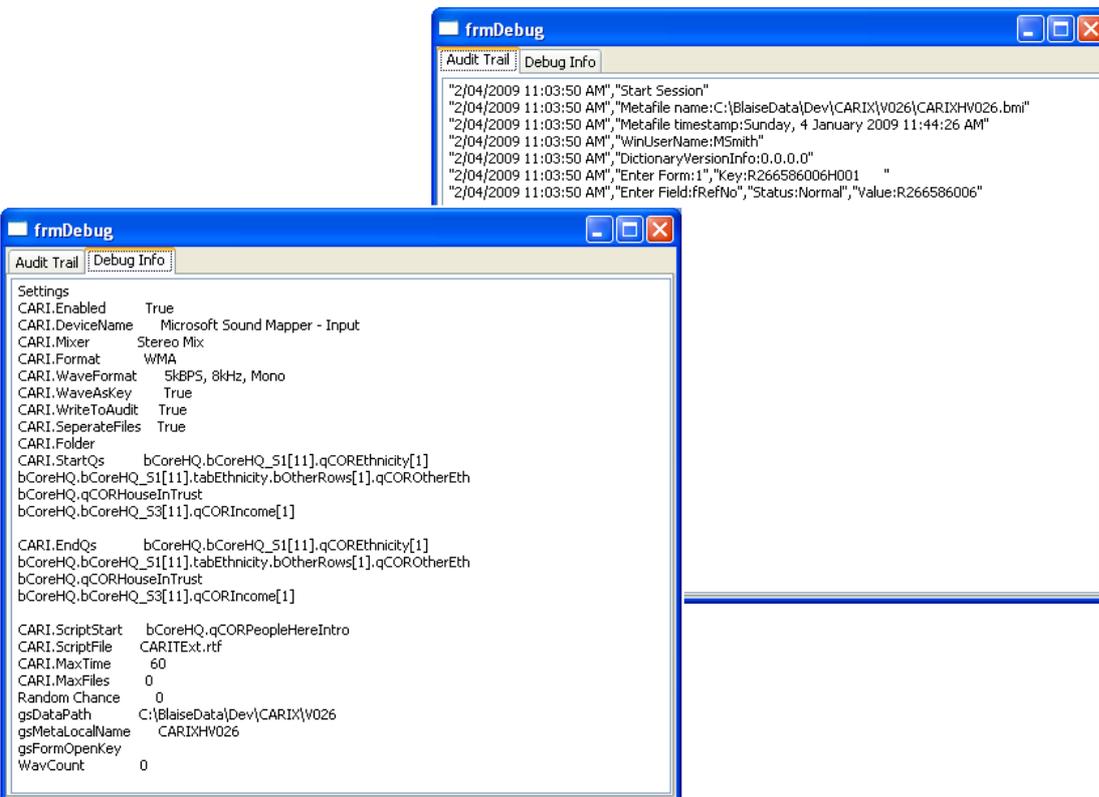
Recording Format

The recording format used to produce the audio files has a significant impact on the usability of the solution, with factors such as hard drive space and transmission bandwidth being important considerations over the longer term.

Through the configuration item 'Format' one of a number of compressed and uncompressed audio formats can be selected to record the audio stream. WMA was selected as our default audio format as it provided a more efficient compression of voice recording and therefore generated smaller files. However any audio codec that is available to the operating system can be used.

Solution Debugging

Another useful feature built into the Statistics New Zealand Audit trial functionality is the ability to monitor the audit information in real time. Enabling the configuration item 'Enabled' in the [Debug] section of the .AIF file causes a separate window to be displayed when the Blaise instrument is started.



Solution Integration

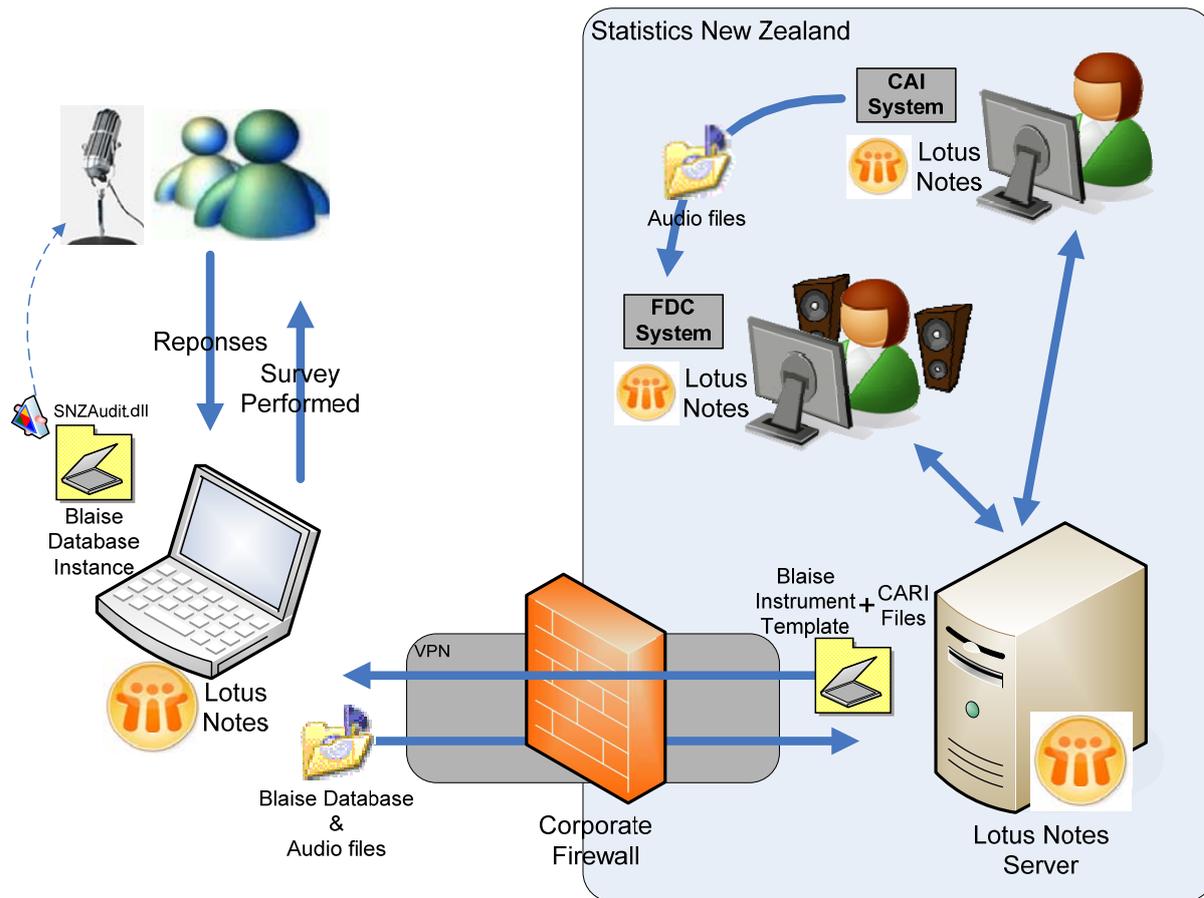
One of the main design goals was to integrate audio recording with Blaise and Statistics New Zealand's field collection applications in a way that caused as little impact and rework as possible to those existing systems.

CARI capture

The CAI system was extended to replicate the CARI configuration file (.AIF file) out to the laptop of the collection agents as the case list and Blaise instrument are transferred.

To enable CARI on a Blaise instrument requires that the instrument is configured to use the SNZAudit.dll as the audit trail library before it is uploaded into the CAI system. This is currently done by the staff responsible for the development of the instrument. Once on the Laptop, security settings prevent respondent from changing any options within the Blaise instrument or CARI. Any CARI related files or audio recordings can not be seen or accessed by the interviewer.

The CAI system was further extended to replicate the audio files created through the interview process back into the organisation along with the completed Blaise instrument and attached back into the CIA system.



CARI Management

Once the audio recordings are received within the organisation, quality assurance staff use the audio recording to rate the interview and interviewer performance in association with other field collection best practices

To assist the review process another application called the Field Collections Directory (FCD) was extended to provide additional functionality that enables quality assurance staff to listen to the audio recording made by CARI and provide a rating or comment on the effectiveness of the interviewer in asking this question.

The FCD also enables questionnaire developers to use CARI recordings to evaluate question performance.

An example of the CARI Case Review screen is shown here. Note the review questions shown below are sample only and can be changed by the Field Collections Manager. The overall question rating is determined from the answers selected by the reviewer.

Interviewer	CAI Demo2	Comments	Overall very well done
Region	Region 0		
Date	26/03/2009 16		
Survey	CARI		
Case ID	R469553004		
Type	<input checked="" type="radio"/> Routine <input type="radio"/> Targeted		
CARI Case Status	Active - Incomplete	Follow-Up Required	<input type="checkbox"/>
Overall Case Rating	80.00%		

1 of 3

Question ID	CARIXR469553004__H027_R469553004H001_bCoreHQ.bCoreHQ_S1[11].qCOREthnicity[1].WMA		
Question text	Which ethnic group or groups do you belong to?		
Answer recorded	Cook Island Maori		
CARI Review Questions	Does the interviewer read the question exactly as worded in entirety, including all alternatives?	<input checked="" type="radio"/> Yes <input type="radio"/> No <input type="radio"/> N/A	Comment
	Does the interviewer record the answer as given?	<input type="radio"/> Yes <input checked="" type="radio"/> No <input type="radio"/> N/A	Comment
	Are the probing guidelines followed?	<input checked="" type="radio"/> Yes <input type="radio"/> No <input type="radio"/> N/A	Comment
	Does the interviewer's response to queries bias the respondents response?	<input type="radio"/> Yes <input type="radio"/> No <input checked="" type="radio"/> N/A	Comment
	Does the interviewer have a slow and distinct speech which allows the respondent to clearly understand the survey questions?	<input type="radio"/> Yes <input type="radio"/> No <input checked="" type="radio"/> N/A	Comment
Question Rating	66.67%		

Evaluating the CARI implementation

Progress

While CARI appears to be a relatively simple concept, the amount of work required to develop and integrate an end-to-end solution should not be underestimated.

In addition to the development of the technical solution a number of other considerations have come to the surface, including:

- Interviewer concerns related to a potential impact on response rates.
- File sizes and the impact on replication – replication takes longer, therefore cost goes up.
- Privacy issues relating to recording respondent responses – ensuring that the audio recordings are secured, both on the interview laptop and within the organisation.
- Procedures to govern and manage the audio recordings over time – Archiving of the audio recording.

While some of these issues have been addressed others have yet to have their impact fully determined as the solution, although complete, has not been tested in the field. Statistics NZ are currently working on identifying an appropriate survey to begin a trial.

It is seen that CARI will meet several critical needs both those specific to field interviewing and in general, CARI provides a means to remotely monitor the quality of the field interview, including adherence to standards during the interview and the reactions of the respondent to survey questions. Statistics New Zealand expects to utilise CARI to assist:

- Evaluating interviewer performance and providing feedback to interviewers
- Identifying questionnaire problems and opportunities for survey improvement
- Detecting interview fabrication and interview errors

Lessons

A change to the operating system on the interviewer laptop's, from Windows XP to Windows Vista, while the CARI solution was being system tested caused a number of issues and ultimately a delay to the intended field test. It is recognised that both the development, system test and production environments must be comprised of the same software to avoid these kinds of problems in the future.

Looking forward it is recognised that there is limited internal knowledge of the development language (Delphi) used to create the CARI application. This poses risk for the on-going support and maintenance of the solution

and it is therefore envisaged that the solution will be redeveloped using the current corporately supported development toolset of Microsoft's .Net framework.

Overall CARI aligns with the Statistics New Zealand strategic goals of statistical excellence, integrity and leadership;

- Statistical excellence - CARI will help ensure that the Field Collections business unit produces relevant and accurate data through improved technical and quality standards.
- Integrity - CARI will improve the transparency of field-based data collection and should lift the level of trust subject matter clients and data users have in the resulting outputs.
- Leadership - CARI has become an accepted quality assurance mechanism internationally and Statistics New Zealand will be leading its adoption in New Zealand.

It is envisioned that CARI will form a key part of the Field Collections' quality assurance programme and is central to the systematic monitoring of data collection in the field and will provide robust, timely information to inform Field Collection decisions and processes now and into the future.

References

Wikipedia, Component Object Model (COM) definition, 2009,

http://en.wikipedia.org/wiki/Component_Object_Model accessed 4/4/2009

Mitchell, CARI Audit Trail Information Sheet, 2008, *Statistics New Zealand internal document*

Hill Christine, Best Practice Field Collections - CARI Software Specification, 2008, *Statistics New Zealand internal document*

Broadhurst Donna, Best Practice Standards (BPS) for Interviewers - CARI Functionality and Requirements, 2008, *Statistics New Zealand internal document*

Broadhurst Donna, Best Practice Standards (BPS) for Interviewers - Quality Assurance Vision Document, 2007, *Statistics New Zealand internal document*

Methods and Informatics Department Statistics Netherlands, Blaise 4.5 Developers Guide, 2002

Development of Survey and Case Management facilities for organisations with minimal survey infrastructure

Fred Wensing, Softscape Solutions, Australia

1. Introduction

In order to conduct a survey you need more than just a questionnaire and a computer to run it on. You also need to have a system which can manage your survey and manage the case records that will be created.

Organisations that run surveys regularly will probably have existing infrastructure which can be used to manage the conduct and operation of those surveys. For them it will be sufficient to modify their interfaces to interact with Blaise questionnaires. For smaller organisations, or ones with limited survey infrastructure, however, it will be necessary to develop some facilities that can provide some survey and case management functions.

This paper describes a basic survey management system that has been developed for use by three organisations that had minimal or no survey infrastructure before choosing to use Blaise for their surveys.

2. Typical situation

The three organisations involved were:

- The Australian Department of Family, Community Services and Indigenous Affairs (Longitudinal Study of Indigenous Children)
- Telethon Institute for Child Health Research
- New Zealand Ministry of Tourism

The typical situation that I found with these three organisations was:

- They want to conduct a survey relevant to their needs
- They want to use computer assisted interviewing
- They have heard of Blaise and obtained a licence (or evaluation licence)
- They undertake to develop or commission a Blaise questionnaire

When they get further down the track, they plan to:

- Purchase notebook computers for the survey
- Employ interviewing staff to conduct the survey

It is not until the field operations were considered in detail, however, that it was realised that some additional infrastructure would be necessary to help manage the survey. In general, these organisations had little or no existing survey infrastructure to assist with the conduct of the survey. The common assumption was that Blaise software could provide these facilities.

There is no ready-made case management facility provided with Blaise software, but the tools for developing such facilities do exist.

3. Choice of tools to develop management facilities

It is possible to develop survey and case management facilities either by using the Maniplus language of the Blaise suite or by using the Blaise API component. In Blaise 4.81 there is also the Blaise Data Centre utility which may have a role.

Maniplus has the advantage that it is part of the Blaise suite and can readily access Blaise data files. The Maniplus language provides relatively easy ways to program dialog elements containing fields and buttons that can be activated or hidden, depending on the situation. The development can be done using the Blaise Control Centre and the skill level is marginally more than needed to develop Blaise questionnaires.

To make use of the Blaise API component it is necessary to develop a separate application using a scripted language (eg. Microsoft Visual Basic, C++ or .NET) that can provide you with an interface which then can be used to interact with Blaise data files. This requires additional development software (eg. Microsoft Visual Studio) as well as suitably skilled programmers to carry out the development. There are also additional licence fees to be able to use the Blaise API component.

The Blaise Data Centre provides an interface to examine Blaise data and carry out some case management functions such as deployment and importation of cases. The facility is interactive and does not seem to provide a programmable interface so that you can tailor it to particular needs. There are also additional licence fees to be able to use the Blaise Data Centre. This utility would have functional use in the office environment and could complement facilities set up on the notebook computer.

It was decided that Maniplus would be used to develop survey management facilities because none of the organisations had ready access other software or skilled programmers. It would also be easier for survey staff, with some skills in Blaise, to maintain a Maniplus application once it was developed. It was also part of the basic Blaise software installation.

4. Facilities needed

When considering the facilities that needed to be developed for the notebook computer it was decided to focus on features that were essential for the direct operation of the survey. General business issues such as secure access to the computer, internet access and communication with the office were considered out of scope because they could be handled by existing software and protocols.

The following features were considered important and necessary to be incorporated into a basic survey and case management system that could operate on the notebook computer:

- Establishment of the survey environment
- Identification of the operator
- Installation and removal of surveys
- Display of current surveys
- Display of cases for the selected survey
- Management of the interview entry and exit
- Managing the status of cases
- Packing up of cases for return to the office

5. System design

The system was designed to have three layers of management programs.

The top layer is the Survey Manager program which provides the single point of entry for the operator to access all surveys on the notebook computer. From within the Survey Manager the operator can install or remove surveys, and/or select one to be used for the current session.

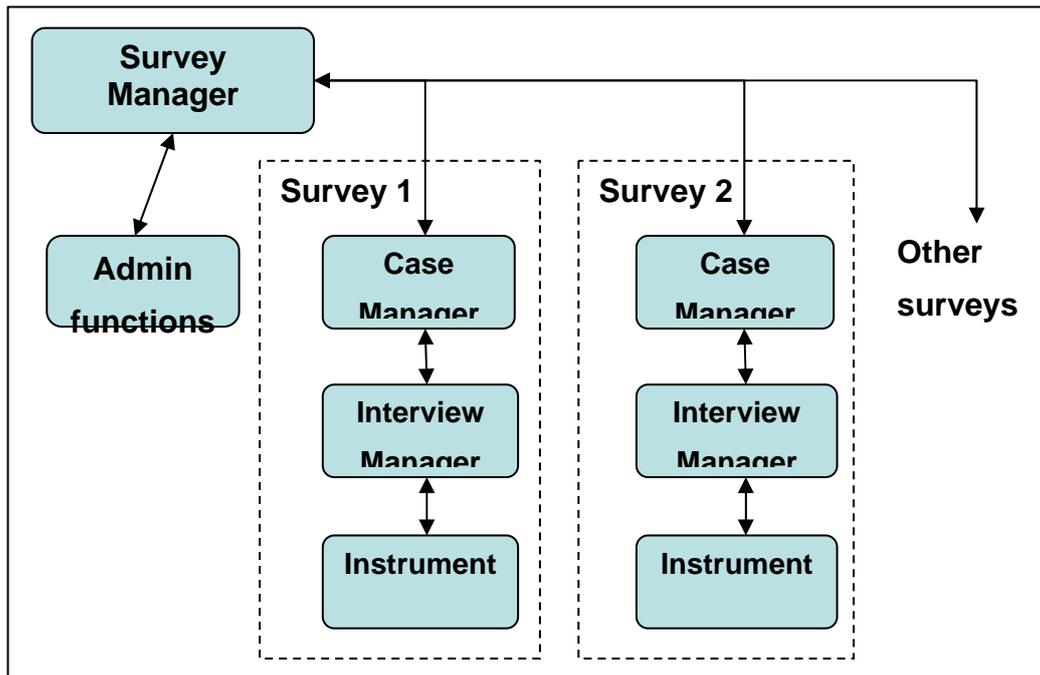
The second layer handles the Case Management functions relevant to each survey. A survey would have its own Maniplus program tailored to its particular needs.

The third layer is the Interview Management layer. Once again, each survey would have its own Maniplus program that handles the entry into and exit from the interview for a selected case record.

Underneath the management layers are the Blaise instruments (electronic questionnaires) that are used in the surveys.

The high level system design is illustrated in Figure 1. Each facility is described in more detail in the sections that follow.

Figure 1. High level system diagram showing three layers of management



6. Survey environment on the notebook computer

The main requirement for the survey environment on the notebook computer is to keep it as simple as possible. That way, it should be relatively easy to resolve any problems that might arise.

6.1 Blaise Software installation

It was decided that the easiest way to set up Blaise on the notebook computers to be used by the interviewers was to install a current complete version of Blaise using the standard Blaise installation package. That way all the necessary software elements, and more, were available on the computer should they be needed.

While it is possible to operate Blaise surveys using a cut-down version of the software (involving a small number of EXE files from the standard installation) that would have required a special installation package to be built. There were no resources to develop or maintain such a special installation package so the full installation was chosen.

It is not necessary to include the Blaise licence key in the installation because the interviewers do not need any of the licensed components.

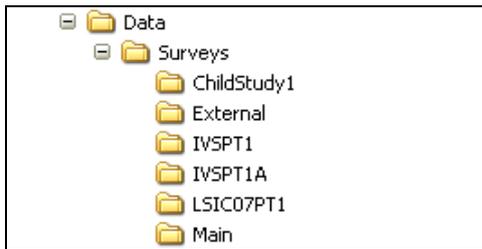
In the event that survey managers are concerned about interviewers running parts of the Blaise software which they should not, it is possible to remove access to the Blaise Control Centre by removing the Blaise.EXE file from the installation. This would have to be done manually.

6.2 Folder structure

In order to manage the surveys to be conducted on the notebook, a simple folder structure was devised.

The primary folder under which the survey manager application and all surveys would be placed was called 'Surveys' and this was placed in a folder on the C drive called 'Data'. The Survey Manager application was placed in a folder called 'Main' and this was located in the 'Surveys' folder. Each survey would then be placed in its own named folder, also in the 'Surveys' folder. In order to share the use of external look-up files across multiple surveys, an 'External' folder may also be useful. See Figure 2 for a screen-capture of the folder structure.

Figure 2. Screen-capture view of survey folders once 4 surveys have been installed



There is no need to set up these folders because the installation processes take care of that.

6.3 Other software

No other software is required to conduct Blaise surveys but it would be expected that some standard software would be available to handle security, internet connection and communications (eg. E-mail client software).

7. Survey Manager application

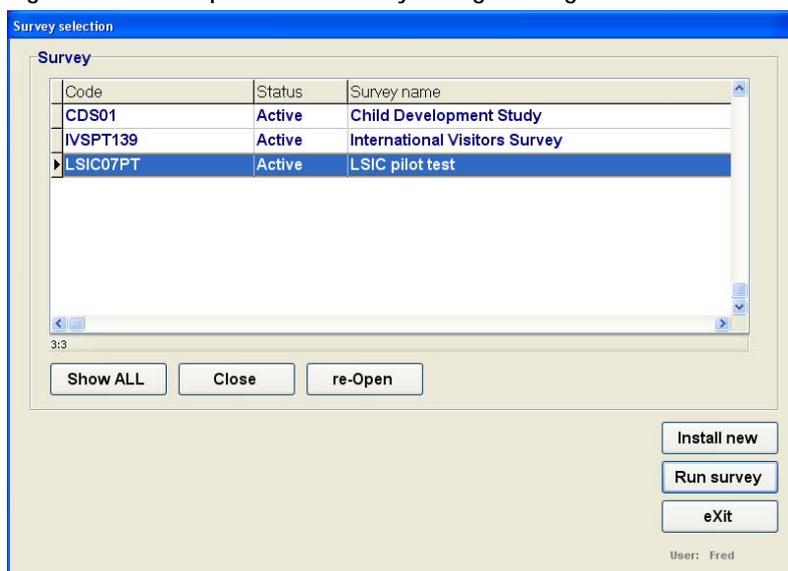
The Survey Manager application provides:

- Single start-up point
- User identification
- Survey/instrument installation
- List of available surveys (for selection)
- Buttons to start a session using a selected survey

7.1 Interface

The interface consists of a screen that lists all available surveys with buttons for installing new surveys and activating current ones (See Figure 3).

Figure 3. Screen-capture of the Survey Manager dialog



7.2 User identification

User identification occurs when the Survey Manager application is run for the first time. At that point a small dialog screen is provided showing a field into which the user can enter his/her user name.

Once a user's identity has been recorded then that is passed down to the other management programs and can eventually be inserted into the case record to identify the operator who has been collecting or working with the data.

It is also possible to assign different roles to different users by storing a list of available users for a survey.

The user identification developed for this application is fairly open. Once a name has been given it is stored and all future uses of the application on that computer assume the same user name. Essentially the application is identifying the computer rather than the operator but this was considered adequate for the needs of the organisations involved.

Access to the Survey Manager application is not restricted in any way as that was considered unnecessary because access to the notebook computer was presumed to be controlled by normal Windows user and password control.

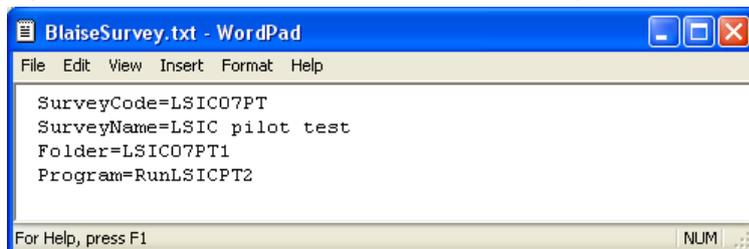
7.3 Installation of a survey through the Survey Manager

The interface provides a button that enables the operator to install a new survey into the system. A new survey would generally be supplied on CD or USB drive which would contain a folder holding all required Blaise instruments, Maniplus programs and any other files needed for the survey.

To control the installation process a small text file called **BlaiseSurvey.TXT** is included with the installation files (see Figure 4). This text file contains the following four items of information

- The full name of the survey
- An identifying code name for the survey
- The target folder name
- The name of the Maniplus case management program to be run

Figure 4. Screen-capture of the content of BlaiseSurvey.TXT



The installation process uses the supplied information to create the target folder and copy the survey files into it. Once the survey files are installed the list of available surveys is updated to show the new survey.

The Survey Manager is set up to prevent the same survey (ie. one with the same Survey identification code) from being installed a second time. This was done to avoid accidentally over-writing collected data.

7.4 List of surveys

The Survey Manager maintains two lists of surveys for the operator. One list contains the names of all surveys ever installed and the other contains the list of active surveys only. The default list is the list of active surveys. Surveys can be moved between the lists using the **Close** and **Re-Open** buttons.

Surveys are never deleted by the system. If a survey actually needs to be deleted then that could be done manually next time the notebook computer is in the office.

7.5 Running a survey from the list

Once a survey is selected then it can be run from the Survey Manager using the **Run Survey** button. When that is done then the system executes a command to the corresponding Maniplus Case Management program, passing in the user name as a parameter. Control is then passed to the Case Management program. When that program finishes then control returns to the Survey Manager.

7.6 Installation of the Survey Manager

The Survey Manager is installed from a CD or USB drive which contains the program and simple Install.BAT file. The install process creates the target folder and copies the program elements.

8. Case Manager applications

The Case Manager application provides:

- Access to a List of cases for the survey
- Ability to operate on the list (load, remove, package)
- Buttons to start an interview using a selected case
- Management of the case records in the survey folder

Each survey generally has a different set of needs and each will therefore require a tailored program. The sections that follow discuss some of the issues and how they can affect the Case Management program.

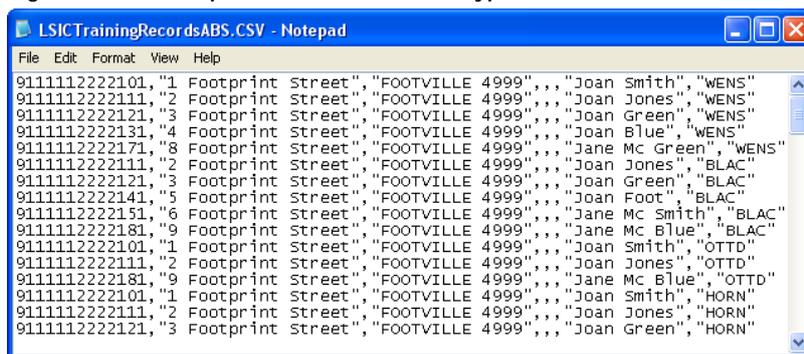
8.1 Pre-defined and allocated list

Many surveys operate on a list of addresses (or cases) which are allocated to particular interviewers.

To manage this kind of survey it is important to maintain a single central list (in the office) which keeps track of the case identifiers and which interviewer has been assigned each case. Whenever a case is moved from one interviewer to another the list needs to be updated in the central repository.

From time to time the updated case list, in the form of a comma-separated text file, would then be sent to the interviewers for loading. Figure 5 shows an example of such a list.

Figure 5. Screen-capture of the content of a typical case list

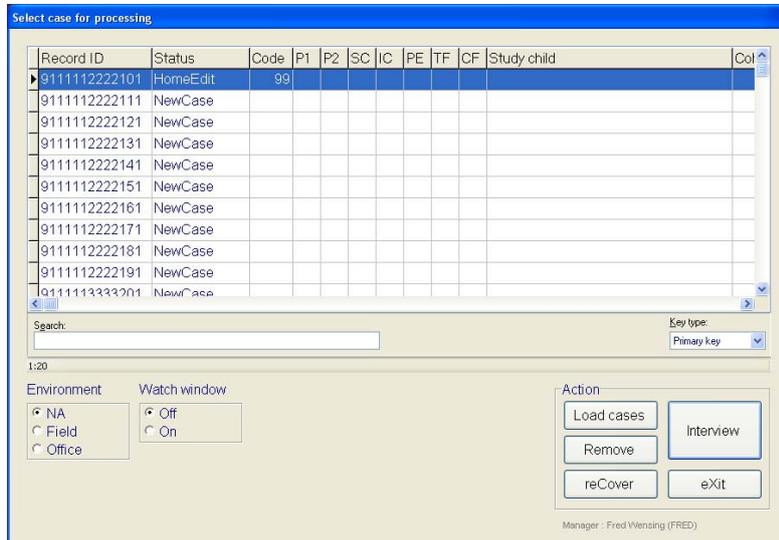


The Case Manager application for such a survey should be able to load the latest case list whenever it is updated. The system also needs to keep track of all the case lists that have been loaded so that it does not re-load an old list.

Where cases are allocated to specific interviewers, the Case Manager would need to select, for display, only those cases that match the user name.

Figure 6 shows a screen-capture of the Case Manager developed for the Survey of Indigenous Children where cases were assigned to particular interviewers. Note the button for loading updated case lists.

Figure 6. Screen-capture of the Case Manager where an assigned list is involved



For convenience of the operator, options exist in the Case Manager interface to remove completed cases from view.

8.2 Dynamic sample

For some surveys, there is no predefined sample. Instead, the sample is defined at the time of interview. This can happen when there is random selection at a venue such as an airport. In this situation the Case Manager needs to create uniquely identified cases.

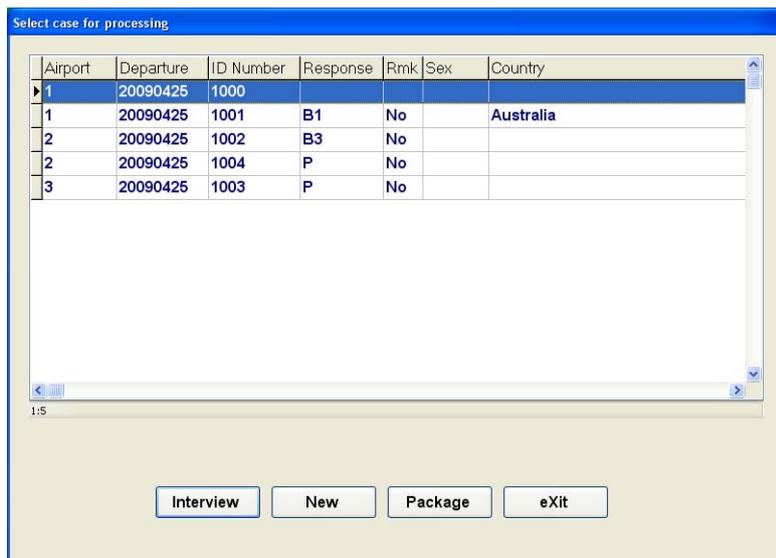
The International Visitor Survey run by the New Zealand Ministry of Tourism is just such a survey. The Case Manager was programmed to set up unique identifiers based on airport, current date and interviewer code. At the commencement of each session the operator is required to identify the airport and date (see Figure 7). The system keeps track of the last used case number and assigns new numbers to new cases.

Figure 7. Screen-capture of the session identification dialog



New cases are added to the case list as the interviewing session progresses. Figure 8 shows a screen-capture of the Case Manager developed for the International Visitor Survey where cases were created dynamically using the **New** button. Previously created cases can be accessed using the **Interview** button.

Figure 8. Screen-capture of the Case Manager for a dynamic sample



8.3 Management of case records in the survey folder

There are two ways to handle the Blaise case records which are created:

- Have a single file containing all case records, or
- Treat each case record as a file in its own right

The single file approach is satisfactory provided that there is no need to move individual cases between interviewers. This was the case with the International Visitor Survey.

However, as soon as there is a possibility that cases may need to be moved around then it is best that each case record be stored in a separate file. This was the situation with the Longitudinal Study of Indigenous Children.

8.3.1 Managing a single file of all records

Using a single Blaise data file to hold all the case records is relatively easy provided that each case has a unique identifier. The Blaise file is set up with the corresponding primary keys for the unique identifier and access to the record is done by using the relevant key.

No special file management is required until the data needs to be sent back to the office. This is discussed in section 8.5.

8.3.2 Managing separate files for each case record

Using separate files for each case record does make the Case Manager programs more complicated to write, but gives the advantage of being able to manage each case separately.

Management of the cases is handled by using the ZIP functions of Blaise (available from version 4.8 onwards) or by using the Winzip command line utilities.

Essentially, once an interview is closed the Case Manager collects all the relevant Blaise files and copies them to a Zip file for storage. If the case record is revisited then the Case Manager removes the Blaise files from the Zip file and places them back into the system before running the Interview Manager program.

The relevant Blaise files are identified using the Blaise database name with the * wildcard to pick up all files. Metadata files are excluded from the Zipping process through the use of a list of exclusions, recorded in a text file called **Exclude.TXT**.

Figure 9 shows the procedure used to Zip the data for one case record (using Blaise commands). In this procedure **IDCode** is the identifier of a case and **aFormName** is the name of the survey instrument (Blaise

datamodel) and **Exclude.TXT** contains a list of files to be excluded from the Zip process. Note that the Audit trail file is also added when present.

Figure 9. Procedure to Zip current data

```
{Procedure to Zip current data file to backup folder using Blaise commands}
PROCEDURE Run_Zip
  DAYFILE('Zip current files to '+IDcode+'.Zip')
  aResult := ZIPFILES ('Backup\'+IDcode+'.zip '+aFormName+'.* /X@Exclude.txt')
  IF ADTEXists=1 THEN
    DAYFILE('Zip ADT file to '+IDcode+'.Zip')
    aResult := ZIPFILES ('Backup\'+IDcode+'.zip '+IDcode+'.adt')
  ENDIF
ENDPROCEDURE
```

Before an interview is started, any existing survey data file is copied to another Zip file called Remove.ZIP then deleted. This is important to ensure that new records are not adversely affected by old data.

Note that unlike the Winzip command line program the ZIPFILES command does not have a 'Move' option that enables removal of the data files but not the datamodel. For that reason the program then deletes all the files then restores the datamodel from the Remove.ZIP file.

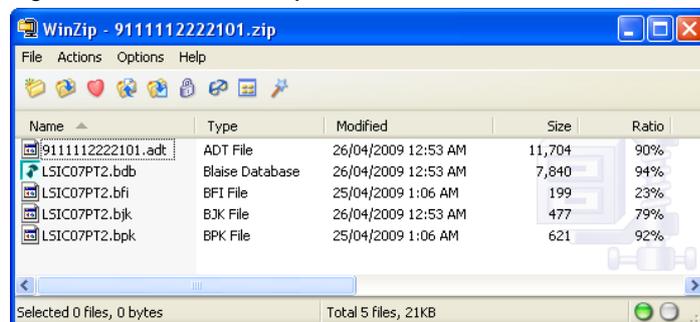
Figure 10 shows the commands that are used to remove any current data and restore a selected case from a Zip file (when it exists). In this part of the program **IDCode** is the identifier of a case, **aFormName** is the name of the survey instrument (Blaise datamodel), **aInstallFolder** is the location of the current survey installation and **Exclude.TXT** contains a list of files to be excluded from the backup Zip process.

Figure 10. Segment of source code showing extraction of case data from backup files

```
{First remove any (old) data files that may be present}
IF FILEEXISTS(aFormName+'.bfi') THEN
  DAYFILE('Clean up files to Remove.zip')
  {Copy the instrument files to a local Zip}
  aResult := ZIPFILES ('Remove.zip '+aFormName+'.*')
  {Delete all survey files}
  aResult := RUN ('DEL '+aFormName+'.*')
  {Extract the survey datamodel from the Remove.Zip file}
  aResult := UNZIPFILES ('Remove.zip /D'+aInstallFolder+' @Exclude.txt')
ENDIF
{If a backup file exists then extract the data and use it}
IF FILEEXISTS('Backup\'+IDcode+'.Zip') THEN
  DAYFILE('Unzip files from '+IDcode+'.Zip')
  aResult := UNZIPFILES ('Backup\'+IDcode+'.Zip /D'+aInstallFolder+' *')
ELSE {otherwise create a new record}
```

Figure 11 shows the content of a single Zip file containing the Blaise data files for one case record and the corresponding Audit trail file. Notice that only the data files are present and that the datamodel is not included.

Figure 11. Content of the Zip file for one case record



8.4 Running the interview(s)

Once a case record has been selected for interview and the data restored (if present) then the Case Manager hands control over to the Interview Manager program, passing in parameters to identify the user and the case.

8.5 Packaging the cases for sending to the office

Depending on the survey, the Case Manager can also be programmed to carry out other case administration tasks such as packaging for return to the office.

The packaging process is a separate Maniplus program that makes use of the ZIP functions of Blaise to put all cases for a designated survey into one Zip file. This makes it easier for the operator to send the data back because the program has made the selection of files and it produces just one file to be attached to an e-mail.

The packaging process is activated through the **Package** button on the Case Manager screen.

Because the packaging process removes those records from the system, the operator is warned that they will no longer have access before execution takes place.

When the packaging is complete a small report is shown on the screen for the operator (see Figure 12).

Figure 12. Screen-capture of the packaging report



9. Interview Manager applications

The Interview Manager application provides:

- Program control over the entry and exit from the interview
- Check of the clean/dirty status of the record and inserts the status value into the data
- Handling of the setting of response status

9.1 Controlling the entry and exit from the interview

The Interview Manager is a Maniplus program that provides a shell around the interview. The program accepts parameters from the Case Manager and then starts up the interview for the identified case using the Manipula EDIT command.

The following command line options are applied to the EDIT command to:

- Identify the case to be opened (option /K)
- Specify the Menu file to be used (option /M)

- Identify whether the Audit trail is to be activated (option /C)
- Turn on the watch window if required (option /!)
- Exit the data entry process when the interview is closed (option /X)

Other options may be added if needed but these are the common ones.

Figure 13 shows a typical activation of the interview using the EDIT command. In this part of the program **aWatchW** is for the watch window setting and **aRecordID** contains the identifier of the case record as supplied via parameters from the Case Manager.

Figure 13. Segment of source code showing the start-up of the interview

```

{Set the watch window based on Parameter 2}
IF PARAMETER(2) = 'ON' THEN
  aWatchW := '/!'
ELSE
  aWatchW := ''
ENDIF

{start-up the interview}
IF PARAMETER(3) <> '' THEN
  {apply the Audit trail settings}
  aResult := fInterview.EDIT('/MIVS.bmf /X '+aWatchW+' /K'+aRecordID
    +' /C'+PARAMETER(3)+' .DIW')
ELSE
  {without the Audit trail}
  aResult := fInterview.EDIT('/MIVS.bmf /X '+aWatchW+' /K'+aRecordID)
ENDIF

```

The most important function that the program provides, however, is to take over when an interview is closed. This is covered in the next two sections.

9.2 Checking the clean/dirty status

Once the interview is closed it is possible for the Interview Manager program to check whether the record is clean or dirty. This is done by examining the FORMSTATUS of the record (see Figure 14). It is then possible to record that status in a field in the record for ease of access by users or other systems.

Figure 14. Segment of source code that checks the clean/dirty status

```

{check the instrument status and record the result in the BlaiseStatus field}
IF fInterview.FORMSTATUS = NOTCHECKED THEN
  fInterview.CHECKRULES
ENDIF
IF fInterview.FORMSTATUS = CLEAN THEN
  fInterview.BlaiseStatus:=BLClean
  aStatus := 'CLEAN' {Instrument status is CLEAN}
ELSEIF fInterview.FORMSTATUS = DIRTY THEN
  fInterview.BlaiseStatus:=BLDirty
  aStatus := 'DIRTY' {Hard errors exist in the instrument}
ELSEIF fInterview.FORMSTATUS = SUSPECT THEN
  fInterview.BlaiseStatus:=BLSuspect
  aStatus := 'SUSPECT' {Soft errors exist in the instrument}
ELSEIF fInterview.FORMSTATUS = NOTCHECKED THEN
  fInterview.BlaiseStatus:=BLNotchecked
  aStatus := 'NOT CHECKED' {Instrument status is NOT CHECKED}
ENDIF

```

9.3 Setting response status

Once the interview is closed the Interview Manager can intervene and provide the operator with a screen where the response status can be set.

The response status options are modified to match the completeness of the interview. The logic for completeness is best done within the Blaise instrument because then it is available to any system that needs this information. Completeness can be tested by placing particular points within the instrument flow then checking how many of those points have been triggered (see Figure 16).

Depending on the completeness of the interview, then only particular response status options are possible. The response status screen is set up so that only the relevant response settings are activated and others are not (see Figure 17). At this point there is also an option for the operator to re-enter the case record if desired. Remarks can also be added in the Response Status dialog and these will be inserted into the case record.

Figure 16. Segment of source code with typical logic for completeness of an interview

```
IF StartInterview=Continue THEN
  IF Endsurvey=Continue THEN
    IF Eligible=No THEN
      StatusFlds.Condition1:=3 (ineligible)
    ELSE
      StatusFlds.Condition1:=1 (full response)
    ENDIF
  ELSE
    StatusFlds.Condition1:=2 (partial response)
  ENDIF
ELSE
  StatusFlds.Condition1:=4 (empty record)
ENDIF
```

Figure 17. Screen capture of the response status dialog with relevant entries activated

Select Response Status

Codes available

- A. Complete interview
- B1. Partial interview, refused**
- B2. Partial interview, boarding**
- B3. Partial interview, other**
- C. Eligible, refused
- D. Eligible, refused entry
- E. Eligible, boarding
- F. Eligible, language
- G. Eligible, other
- H. Unknown, refused
- I. Unknown, boarding
- J. Unknown, language
- K. Unknown, other
- L. Missed, no interviewer available
- M. Missed, other
- N. Ineligible (Out of scope)
- O. Quota full
- P. Other

Response remarks:

Initial response status: B1

Re-enter eXit

Information
No remarks in this interview.

Upon completion of the Response Status dialog, control is returned to the Case Manager.

10. Summary

The programs described in this paper illustrate the kind of facilities that are needed to manage one or more surveys that involve the use of notebook computers in the field. It shows how there are three layers of management that need to be supported: survey management, case management and interview management.

For organisations that have little or no existing survey infrastructure, Maniplus clearly has sufficient functionality and versatility to provide the solution.

Case Management System Based on Wireless Telecommunications

Vesa Kuusela, Toni Räikkönen and Kai Vikki, Statistics Finland

Introduction

At Statistics Finland the CAPI system was introduced in 1993 for the use of the Social Survey Unit (see Kuusela, 1995). Since then the system has been renovated several times. The most important change was in mid 90s when the Case Management System (CMS) was renewed from trivial (all interviewers had the whole sample) to an object based system (see Kuusela, 1997). The basic structure of the implemented CMS has proved to be reliable and enduring. The next major change took place at the end of millennium when user interfaces were designed anew applying the possibilities provided by the modern Windows tools (see Kuusela, 2001). The CMS of Statistics Finland CAPI system was renewed in 2008 - 2009 and it is currently operational.

In the previous system data exchange between field interviewers and office was based on traditional fixed telephone lines and modems. Modern telecommunications facilities provide new possibilities which did not exist few years ago. The new system is based on wireless telecommunications, Wireless Wide Area Network (WWAN). The WWAN system for data transmission makes use of mobile "modems" which connect a (laptop) computer to the Internet with a broadband type connection.

Technical set-up

Currently wireless 3G and UMTS networks¹ have wide coverage in most countries. They are mainly designed for the use of new mobile telephones but their use as a wireless connection to the internet is increasing rapidly. In Finland, for example, 3G/UMTS network covers the greatest part of country and GPRS² covers the whole country. Data transmission with UMTS network reaches the speed of fast wire broadband. For the needs of a CAPI information system GPRS is fast enough. The fast development of telecommunications technology brings up new and fascinating possibilities for designing a CAPI Case Management System (CMS).

The system which Statistics Finland took in use includes transfer rate of downstream on 5 Mbit and 2 Mbit upstream. This service has a fixed monthly price per user, independent of the amount of data transferred.

In Figure 1 is shown the technical structure of the new CMS. Communication is based on a secure file transfer protocol (SFTP). The SFTP allows file transfers over SSH using traditional FTP commands for downloading and uploading files. The communications is based on the idea of using objects. The system architecture was presented in the 4th IBUC (see Kuusela and Parviainen, 1997). All data that is transferred between interviewers and the office system is compressed in a single file called *communications object*. Compressing is used only for encapsulation. The reduction of file size is not important, anymore. A communication session initiated by an interviewer laptop may handle several objects, such as interviews from several different surveys, case transfers, pay claims, etc.

¹ Universal Mobile Telecommunications System (UMTS) is one of the third-generation (3G) mobile telecommunications technologies. Currently, the most common form of UMTS uses W-CDMA (Wideband Code Division Multiple Access). W-CDMA is a high speed transmission protocol designed for mobile telecommunications networks. In theory, UMTS, using W-CDMA, supports up to 21 Mbit/s data transfer rates. In practice, transfer rates are much lower. UMTS is sometimes also called 3GSM to emphasizing the combination of the 3G technology and the GSM standard.

² General packet radio service (GPRS) is a mobile data service available to users of the 2G mobile communication systems for GSM and 3G networks. In the 2G systems, GPRS provides data transfer rates up to 114 kbit/s.

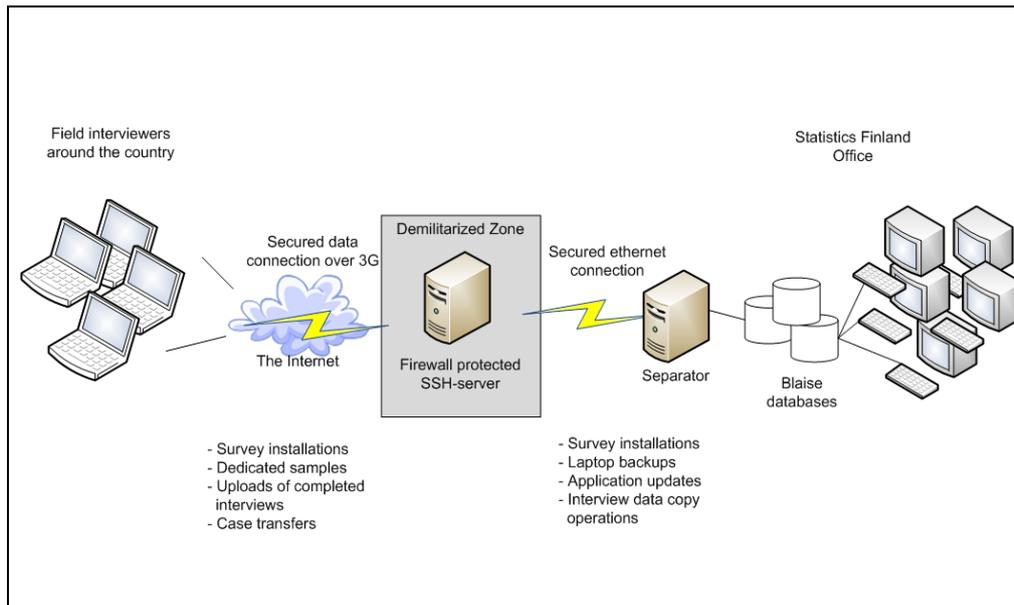


Figure 1: Technical design of the CAPI information system.

The FTP server at Statistics Finland is firewall protected on the demilitarized zone. In the office intranet, a dedicated server called *Separator* polls the FTP server, downloads the incoming files and extracts the data from communication objects to appropriate places. Survey specific data are placed on appropriate survey folders, most of the case transfers from one interviewer to another are directed automatically by *Separator* (some go to supervisors).

Case Management System design

At Statistics Finland an important background factor for the system design is the fact that the Social Survey Units undertake more than 30 different surveys each year, one of which is the monthly Labor Force Survey. A great number of these surveys are commissioned by clients coming out side of Statistics Finland and the clients pay a market price of this service. Occasionally the commissioned surveys are done with a very tight schedule which puts some pressure on the reliability and easy maneuverability of the system. Occasionally a survey has to be installed within few days and results have to be ready on a given date.

At the same time with introduction of the new techniques also the software was designed anew. The existing concept had served well the operations of the field unit. Therefore, the basic architecture of the new information system is close to the previous one. In the core there is object based design (see Kuusela, 1995).

The most visible change took place in the user interfaces which were designed anew. The previous functionality was retained but implemented in a different manner. In addition, the modern telecommunications and the Internet provide some fascinating new possibilities, such as possibility to view road routes and driving directions from the Internet map services, such as the Google Maps.

The use of fast telecommunications makes the design of some parts of the CMS more simple than it was earlier. A special feature in Finland is that samples are drawn from an updated Population Register and consequently the sample files include accurate contact information (e.g. names, address, etc.), in addition to some personal information (e.g. age, gender, education, etc.). Telephone numbers are searched from the telephone register. In the previous CMS, a sample was divided to interviewers by an application in the CAPI office system, and after that the dedicated samples were delivered to each interviewer. The main reason was to keep the amount of transmitted data as low as possible.

In the new system, the whole sample file is sent to each interviewer. In the sample file, each record also includes a field indicating the interviewer whom the case belongs to. This simplifies the case transfers essentially: it suffices only to change this field in both interviewer's sample file and the only information transferred is the identification of the survey and respondent id within this survey. Sample file (as nearly all other files) is in XML format. In the interviewer's interface on the laptop, only a view to the sample file (for the specific interviewer) is shown.

In the FTP server, there is an *inbox* and *outbox* for each interviewer. When interviewer opens telecommunications application, it sends automatically all data from outbox of the laptop to the inbox of the server. After that all data in the outbox of the server is copied to the inbox of the laptop. Everything happens automatically and interviewer only initiates the telecommunications process.

Computer as a telephone

The field interviewers of Statistics Finland also collect data by decentralized CATI. Therefore, each interviewer has a landline telephone paid by Statistics Finland. The new wireless application for data transfer includes also a voice client ("telephone"). Field interviewers are now able to use the laptop with a headset as a telephone.

Telephone client uses "normal" GSM network. i.e. calls are normal mobile telephone calls, not VOIP calls. Calls are charged by the same tariffs as the mobile phone calls. In Finland, the most expensive calls are from a fixed telephone to a mobile phone and calls between two mobile phones are only slightly more expensive than calls between two fixed telephones. Nearly 80% of interviews in Finland are done with respondent using a mobile phone. If interviewers call using mobile phone or the computer using mobile phone network, considerable saving may be obtained.

Discussion

The new technical set-up is based on new technical solutions and its introduction went smoothly. Data exchange is fast in nearly all cases. Only few interviewers get only GPRS type connection which is slow, but even that is considerable faster than the old modem connection. The experience so far has shown that the wireless telecommunications system is as reliable as the one using fixed lines.

In addition to the enhancements in Case Management System, few other new applications could be installed. Interviewers are now able to view the general information pages in the Statistics Finland intranet, and they are now connected as clients to the Statistics Finland email system. Both of these are using a VPN connection. Standard access to the Internet is naturally also possible. Interviewers actually use it a lot for example to search telephone numbers from public telephone registry.

Price of the telecommunication per user remains the same independent of the use. This fact brings along predictability on budgeting and it is expected that the total costs of data collection will drop considerably in the future. If the computer telephone client proves to be usable, the savings will be even more.

The computer works independently from fixed telephone lines everywhere where exists an operational mobile telephone (GSM) network. For the future development this provides exciting possibilities for CMS design. For example, it is possible to design a CMS which is based on a CATI system type of design. Actually, technically it would be possible at the moment but it requires careful consideration for the management point of view before it can be implemented.

References

- Kuusela V. (1995) Interviewer Interface of the CAPI-system of Statistics Finland. In Kuusela V. (ed.): *Essays on Blaise 1995*. Statistics Finland, 1995.
- Kuusela, V., Parviainen, A. (1997). An Object-Oriented Case Management System for CAPI Surveys. *Actes de la 4^e Conférence Internationale des Utilisateurs de Blaise*. INSEE, 1997.
- Kuusela, V. (2001): A Windows Based User Interface and CAPI Information System. *Proceedings of the seventh Blaise Users Conference*. Washington, 2001.

Development of Survey and Case Management facilities for organisations with minimal survey infrastructure

Fred Wensing, Softscape Solutions, Australia

1. Introduction

In order to conduct a survey you need more than just a questionnaire and a computer to run it on. You also need to have a system which can manage your survey and manage the case records that will be created.

Organisations that run surveys regularly will probably have existing infrastructure which can be used to manage the conduct and operation of those surveys. For them it will be sufficient to modify their interfaces to interact with Blaise questionnaires. For smaller organisations, or ones with limited survey infrastructure, however, it will be necessary to develop some facilities that can provide some survey and case management functions.

This paper describes a basic survey management system that has been developed for use by three organisations that had minimal or no survey infrastructure before choosing to use Blaise for their surveys.

2. Typical situation

The three organisations involved were:

- The Australian Department of Family, Community Services and Indigenous Affairs (Longitudinal Study of Indigenous Children)
- Telethon Institute for Child Health Research
- New Zealand Ministry of Tourism

The typical situation that I found with these three organisations was:

- They want to conduct a survey relevant to their needs
- They want to use computer assisted interviewing
- They have heard of Blaise and obtained a licence (or evaluation licence)
- They undertake to develop or commission a Blaise questionnaire

When they get further down the track, they plan to:

- Purchase notebook computers for the survey
- Employ interviewing staff to conduct the survey

It is not until the field operations were considered in detail, however, that it was realised that some additional infrastructure would be necessary to help manage the survey. In general, these organisations had little or no existing survey infrastructure to assist with the conduct of the survey. The common assumption was that Blaise software could provide these facilities.

There is no ready-made case management facility provided with Blaise software, but the tools for developing such facilities do exist.

3. Choice of tools to develop management facilities

It is possible to develop survey and case management facilities either by using the Maniplus language of the Blaise suite or by using the Blaise API component. In Blaise 4.81 there is also the Blaise Data Centre utility which may have a role.

Maniplus has the advantage that it is part of the Blaise suite and can readily access Blaise data files. The Maniplus language provides relatively easy ways to program dialog elements containing fields and buttons that can be activated or hidden, depending on the situation. The development can be done using the Blaise Control Centre and the skill level is marginally more than needed to develop Blaise questionnaires.

To make use of the Blaise API component it is necessary to develop a separate application using a scripted language (eg. Microsoft Visual Basic, C++ or .NET) that can provide you with an interface which then can be used to interact with Blaise data files. This requires additional development software (eg. Microsoft Visual Studio) as well as suitably skilled programmers to carry out the development. There are also additional licence fees to be able to use the Blaise API component.

The Blaise Data Centre provides an interface to examine Blaise data and carry out some case management functions such as deployment and importation of cases. The facility is interactive and does not seem to provide a programmable interface so that you can tailor it to particular needs. There are also additional licence fees to be able to use the Blaise Data Centre. This utility would have functional use in the office environment and could complement facilities set up on the notebook computer.

It was decided that Maniplus would be used to develop survey management facilities because none of the organisations had ready access other software or skilled programmers. It would also be easier for survey staff, with some skills in Blaise, to maintain a Maniplus application once it was developed. It was also part of the basic Blaise software installation.

4. Facilities needed

When considering the facilities that needed to be developed for the notebook computer it was decided to focus on features that were essential for the direct operation of the survey. General business issues such as secure access to the computer, internet access and communication with the office were considered out of scope because they could be handled by existing software and protocols.

The following features were considered important and necessary to be incorporated into a basic survey and case management system that could operate on the notebook computer:

- Establishment of the survey environment
- Identification of the operator
- Installation and removal of surveys
- Display of current surveys
- Display of cases for the selected survey
- Management of the interview entry and exit
- Managing the status of cases
- Packing up of cases for return to the office

5. System design

The system was designed to have three layers of management programs.

The top layer is the Survey Manager program which provides the single point of entry for the operator to access all surveys on the notebook computer. From within the Survey Manager the operator can install or remove surveys, and/or select one to be used for the current session.

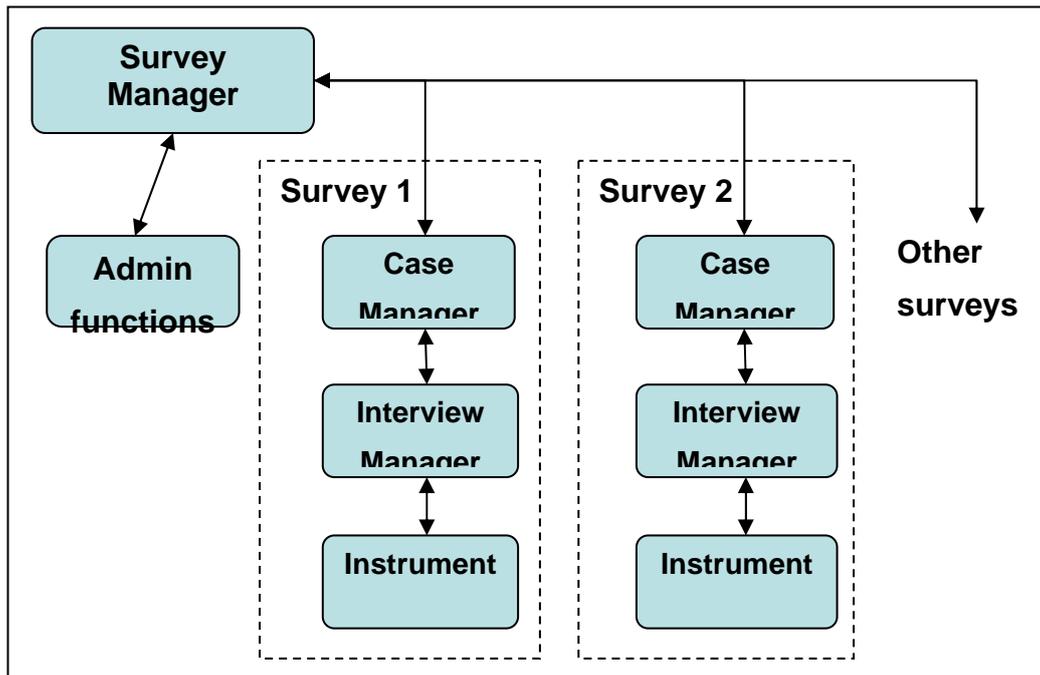
The second layer handles the Case Management functions relevant to each survey. A survey would have its own Maniplus program tailored to its particular needs.

The third layer is the Interview Management layer. Once again, each survey would have its own Maniplus program that handles the entry into and exit from the interview for a selected case record.

Underneath the management layers are the Blaise instruments (electronic questionnaires) that are used in the surveys.

The high level system design is illustrated in Figure 1. Each facility is described in more detail in the sections that follow.

Figure 1. High level system diagram showing three layers of management



6. Survey environment on the notebook computer

The main requirement for the survey environment on the notebook computer is to keep it as simple as possible. That way, it should be relatively easy to resolve any problems that might arise.

6.1 Blaise Software installation

It was decided that the easiest way to set up Blaise on the notebook computers to be used by the interviewers was to install a current complete version of Blaise using the standard Blaise installation package. That way all the necessary software elements, and more, were available on the computer should they be needed.

While it is possible to operate Blaise surveys using a cut-down version of the software (involving a small number of EXE files from the standard installation) that would have required a special installation package to be built. There were no resources to develop or maintain such a special installation package so the full installation was chosen.

It is not necessary to include the Blaise licence key in the installation because the interviewers do not need any of the licensed components.

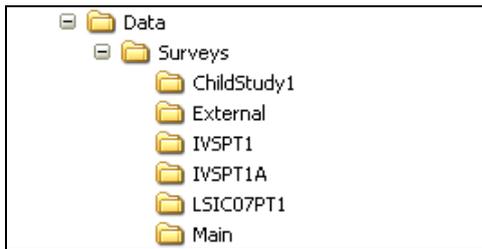
In the event that survey managers are concerned about interviewers running parts of the Blaise software which they should not, it is possible to remove access to the Blaise Control Centre by removing the Blaise.EXE file from the installation. This would have to be done manually.

6.2 Folder structure

In order to manage the surveys to be conducted on the notebook, a simple folder structure was devised.

The primary folder under which the survey manager application and all surveys would be placed was called 'Surveys' and this was placed in a folder on the C drive called 'Data'. The Survey Manager application was placed in a folder called 'Main' and this was located in the 'Surveys' folder. Each survey would then be placed in its own named folder, also in the 'Surveys' folder. In order to share the use of external look-up files across multiple surveys, an 'External' folder may also be useful. See Figure 2 for a screen-capture of the folder structure.

Figure 2. Screen-capture view of survey folders once 4 surveys have been installed



There is no need to set up these folders because the installation processes take care of that.

6.3 Other software

No other software is required to conduct Blaise surveys but it would be expected that some standard software would be available to handle security, internet connection and communications (eg. E-mail client software).

7. Survey Manager application

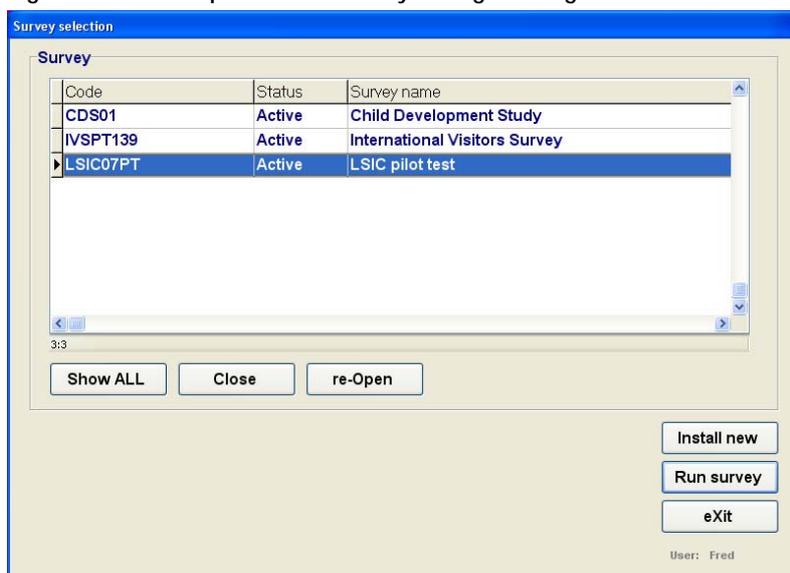
The Survey Manager application provides:

- Single start-up point
- User identification
- Survey/instrument installation
- List of available surveys (for selection)
- Buttons to start a session using a selected survey

7.1 Interface

The interface consists of a screen that lists all available surveys with buttons for installing new surveys and activating current ones (See Figure 3).

Figure 3. Screen-capture of the Survey Manager dialog



7.2 User identification

User identification occurs when the Survey Manager application is run for the first time. At that point a small dialog screen is provided showing a field into which the user can enter his/her user name.

Once a user's identity has been recorded then that is passed down to the other management programs and can eventually be inserted into the case record to identify the operator who has been collecting or working with the data.

It is also possible to assign different roles to different users by storing a list of available users for a survey.

The user identification developed for this application is fairly open. Once a name has been given it is stored and all future uses of the application on that computer assume the same user name. Essentially the application is identifying the computer rather than the operator but this was considered adequate for the needs of the organisations involved.

Access to the Survey Manager application is not restricted in any way as that was considered unnecessary because access to the notebook computer was presumed to be controlled by normal Windows user and password control.

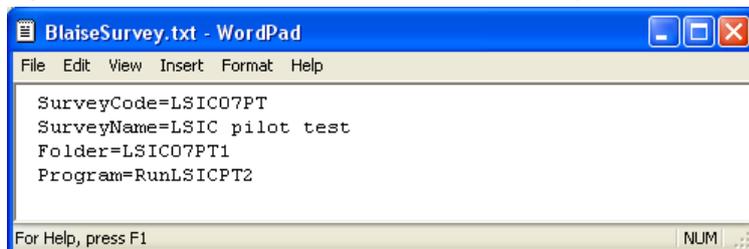
7.3 Installation of a survey through the Survey Manager

The interface provides a button that enables the operator to install a new survey into the system. A new survey would generally be supplied on CD or USB drive which would contain a folder holding all required Blaise instruments, Maniplus programs and any other files needed for the survey.

To control the installation process a small text file called **BlaiseSurvey.TXT** is included with the installation files (see Figure 4). This text file contains the following four items of information

- The full name of the survey
- An identifying code name for the survey
- The target folder name
- The name of the Maniplus case management program to be run

Figure 4. Screen-capture of the content of BlaiseSurvey.TXT



The installation process uses the supplied information to create the target folder and copy the survey files into it. Once the survey files are installed the list of available surveys is updated to show the new survey.

The Survey Manager is set up to prevent the same survey (ie. one with the same Survey identification code) from being installed a second time. This was done to avoid accidentally over-writing collected data.

7.4 List of surveys

The Survey Manager maintains two lists of surveys for the operator. One list contains the names of all surveys ever installed and the other contains the list of active surveys only. The default list is the list of active surveys. Surveys can be moved between the lists using the **Close** and **Re-Open** buttons.

Surveys are never deleted by the system. If a survey actually needs to be deleted then that could be done manually next time the notebook computer is in the office.

7.5 Running a survey from the list

Once a survey is selected then it can be run from the Survey Manager using the **Run Survey** button. When that is done then the system executes a command to the corresponding Maniplus Case Management program, passing in the user name as a parameter. Control is then passed to the Case Management program. When that program finishes then control returns to the Survey Manager.

7.6 Installation of the Survey Manager

The Survey Manager is installed from a CD or USB drive which contains the program and simple Install.BAT file. The install process creates the target folder and copies the program elements.

8. Case Manager applications

The Case Manager application provides:

- Access to a List of cases for the survey
- Ability to operate on the list (load, remove, package)
- Buttons to start an interview using a selected case
- Management of the case records in the survey folder

Each survey generally has a different set of needs and each will therefore require a tailored program. The sections that follow discuss some of the issues and how they can affect the Case Management program.

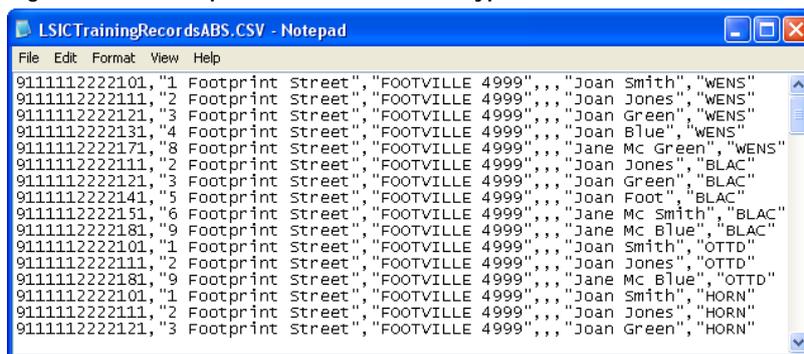
8.1 Pre-defined and allocated list

Many surveys operate on a list of addresses (or cases) which are allocated to particular interviewers.

To manage this kind of survey it is important to maintain a single central list (in the office) which keeps track of the case identifiers and which interviewer has been assigned each case. Whenever a case is moved from one interviewer to another the list needs to be updated in the central repository.

From time to time the updated case list, in the form of a comma-separated text file, would then be sent to the interviewers for loading. Figure 5 shows an example of such a list.

Figure 5. Screen-capture of the content of a typical case list

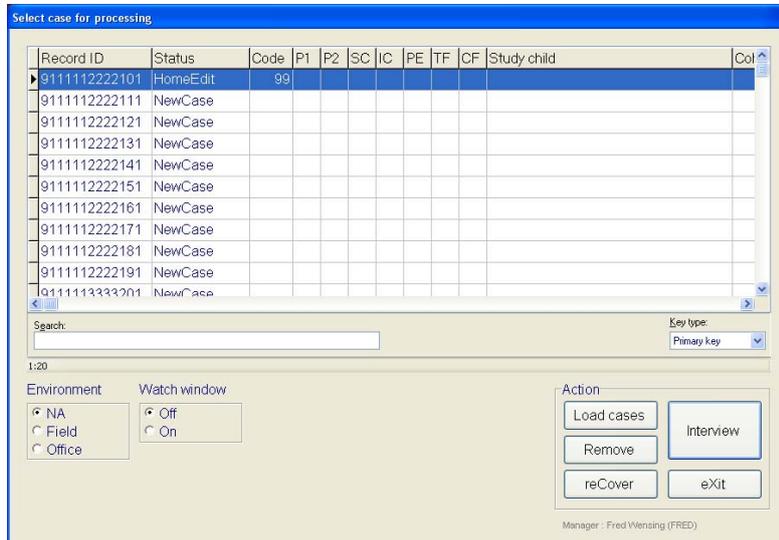


The Case Manager application for such a survey should be able to load the latest case list whenever it is updated. The system also needs to keep track of all the case lists that have been loaded so that it does not re-load an old list.

Where cases are allocated to specific interviewers, the Case Manager would need to select, for display, only those cases that match the user name.

Figure 6 shows a screen-capture of the Case Manager developed for the Survey of Indigenous Children where cases were assigned to particular interviewers. Note the button for loading updated case lists.

Figure 6. Screen-capture of the Case Manager where an assigned list is involved



For convenience of the operator, options exist in the Case Manager interface to remove completed cases from view.

8.2 Dynamic sample

For some surveys, there is no predefined sample. Instead, the sample is defined at the time of interview. This can happen when there is random selection at a venue such as an airport. In this situation the Case Manager needs to create uniquely identified cases.

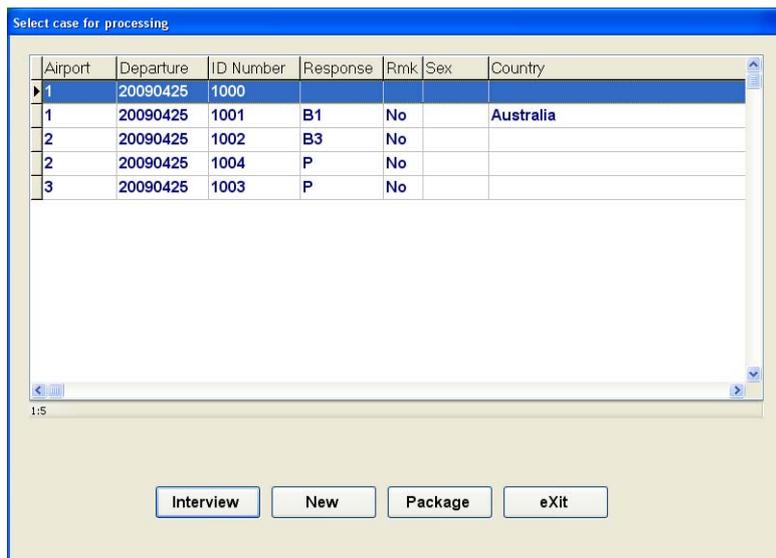
The International Visitor Survey run by the New Zealand Ministry of Tourism is just such a survey. The Case Manager was programmed to set up unique identifiers based on airport, current date and interviewer code. At the commencement of each session the operator is required to identify the airport and date (see Figure 7). The system keeps track of the last used case number and assigns new numbers to new cases.

Figure 7. Screen-capture of the session identification dialog



New cases are added to the case list as the interviewing session progresses. Figure 8 shows a screen-capture of the Case Manager developed for the International Visitor Survey where cases were created dynamically using the **New** button. Previously created cases can be accessed using the **Interview** button.

Figure 8. Screen-capture of the Case Manager for a dynamic sample



8.3 Management of case records in the survey folder

There are two ways to handle the Blaise case records which are created:

- Have a single file containing all case records, or
- Treat each case record as a file in its own right

The single file approach is satisfactory provided that there is no need to move individual cases between interviewers. This was the case with the International Visitor Survey.

However, as soon as there is a possibility that cases may need to be moved around then it is best that each case record be stored in a separate file. This was the situation with the Longitudinal Study of Indigenous Children.

8.3.1 Managing a single file of all records

Using a single Blaise data file to hold all the case records is relatively easy provided that each case has a unique identifier. The Blaise file is set up with the corresponding primary keys for the unique identifier and access to the record is done by using the relevant key.

No special file management is required until the data needs to be sent back to the office. This is discussed in section 8.5.

8.3.2 Managing separate files for each case record

Using separate files for each case record does make the Case Manager programs more complicated to write, but gives the advantage of being able to manage each case separately.

Management of the cases is handled by using the ZIP functions of Blaise (available from version 4.8 onwards) or by using the Winzip command line utilities.

Essentially, once an interview is closed the Case Manager collects all the relevant Blaise files and copies them to a Zip file for storage. If the case record is revisited then the Case Manager removes the Blaise files from the Zip file and places them back into the system before running the Interview Manager program.

The relevant Blaise files are identified using the Blaise database name with the * wildcard to pick up all files. Metadata files are excluded from the Zipping process through the use of a list of exclusions, recorded in a text file called **Exclude.TXT**.

Figure 9 shows the procedure used to Zip the data for one case record (using Blaise commands). In this procedure **IDCode** is the identifier of a case and **aFormName** is the name of the survey instrument (Blaise

datamodel) and **Exclude.TXT** contains a list of files to be excluded from the Zip process. Note that the Audit trail file is also added when present.

Figure 9. Procedure to Zip current data

```
{Procedure to Zip current data file to backup folder using Blaise commands}
PROCEDURE Run_Zip
  DAYFILE('Zip current files to '+IDcode+'.Zip')
  aResult := ZIPFILES ('Backup\'+IDcode+'.zip '+aFormName+'.* /X@Exclude.txt')
  IF ADTEXists=1 THEN
    DAYFILE('Zip ADT file to '+IDcode+'.Zip')
    aResult := ZIPFILES ('Backup\'+IDcode+'.zip '+IDcode+'.adt')
  ENDIF
ENDPROCEDURE
```

Before an interview is started, any existing survey data file is copied to another Zip file called Remove.ZIP then deleted. This is important to ensure that new records are not adversely affected by old data.

Note that unlike the Winzip command line program the ZIPFILES command does not have a 'Move' option that enables removal of the data files but not the datamodel. For that reason the program then deletes all the files then restores the datamodel from the Remove.ZIP file.

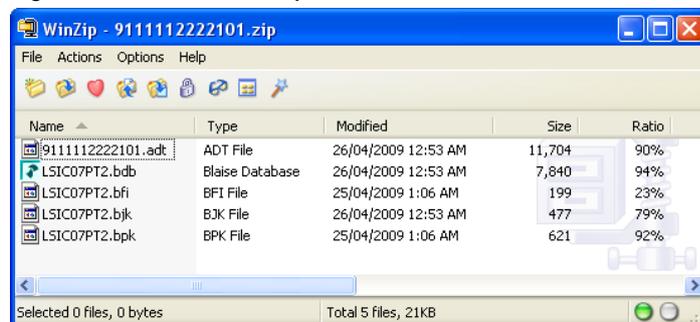
Figure 10 shows the commands that are used to remove any current data and restore a selected case from a Zip file (when it exists). In this part of the program **IDCode** is the identifier of a case, **aFormName** is the name of the survey instrument (Blaise datamodel), **aInstallFolder** is the location of the current survey installation and **Exclude.TXT** contains a list of files to be excluded from the backup Zip process.

Figure 10. Segment of source code showing extraction of case data from backup files

```
{First remove any (old) data files that may be present}
IF FILEEXISTS(aFormName+'.bfi') THEN
  DAYFILE('Clean up files to Remove.zip')
  {Copy the instrument files to a local Zip}
  aResult := ZIPFILES ('Remove.zip '+aFormName+'.*')
  {Delete all survey files}
  aResult := RUN ('DEL '+aFormName+'.*')
  {Extract the survey datamodel from the Remove.Zip file}
  aResult := UNZIPFILES ('Remove.zip /D'+aInstallFolder+' @Exclude.txt')
ENDIF
{If a backup file exists then extract the data and use it}
IF FILEEXISTS('Backup\'+IDcode+'.Zip') THEN
  DAYFILE('Unzip files from '+IDcode+'.Zip')
  aResult := UNZIPFILES ('Backup\'+IDcode+'.Zip /D'+aInstallFolder+' *')
ELSE {otherwise create a new record}
```

Figure 11 shows the content of a single Zip file containing the Blaise data files for one case record and the corresponding Audit trail file. Notice that only the data files are present and that the datamodel is not included.

Figure 11. Content of the Zip file for one case record



8.4 Running the interview(s)

Once a case record has been selected for interview and the data restored (if present) then the Case Manager hands control over to the Interview Manager program, passing in parameters to identify the user and the case.

8.5 Packaging the cases for sending to the office

Depending on the survey, the Case Manager can also be programmed to carry out other case administration tasks such as packaging for return to the office.

The packaging process is a separate Maniplus program that makes use of the ZIP functions of Blaise to put all cases for a designated survey into one Zip file. This makes it easier for the operator to send the data back because the program has made the selection of files and it produces just one file to be attached to an e-mail.

The packaging process is activated through the **Package** button on the Case Manager screen.

Because the packaging process removes those records from the system, the operator is warned that they will no longer have access before execution takes place.

When the packaging is complete a small report is shown on the screen for the operator (see Figure 12).

Figure 12. Screen-capture of the packaging report



9. Interview Manager applications

The Interview Manager application provides:

- Program control over the entry and exit from the interview
- Check of the clean/dirty status of the record and inserts the status value into the data
- Handling of the setting of response status

9.1 Controlling the entry and exit from the interview

The Interview Manager is a Maniplus program that provides a shell around the interview. The program accepts parameters from the Case Manager and then starts up the interview for the identified case using the Manipula EDIT command.

The following command line options are applied to the EDIT command to:

- Identify the case to be opened (option /K)
- Specify the Menu file to be used (option /M)

- Identify whether the Audit trail is to be activated (option /C)
- Turn on the watch window if required (option /!)
- Exit the data entry process when the interview is closed (option /X)

Other options may be added if needed but these are the common ones.

Figure 13 shows a typical activation of the interview using the EDIT command. In this part of the program **aWatchW** is for the watch window setting and **aRecordID** contains the identifier of the case record as supplied via parameters from the Case Manager.

Figure 13. Segment of source code showing the start-up of the interview

```

{Set the watch window based on Parameter 2}
IF PARAMETER(2) = 'ON' THEN
  aWatchW := '/!'
ELSE
  aWatchW := ' '
ENDIF

{start-up the interview}
IF PARAMETER(3) <> ' ' THEN
  {apply the Audit trail settings}
  aResult := fInterview.EDIT('/MIVS.bmf /X '+aWatchW+' /K'+aRecordID
    +' /C'+PARAMETER(3)+' .DIW')
ELSE
  {without the Audit trail}
  aResult := fInterview.EDIT('/MIVS.bmf /X '+aWatchW+' /K'+aRecordID)
ENDIF

```

The most important function that the program provides, however, is to take over when an interview is closed. This is covered in the next two sections.

9.2 Checking the clean/dirty status

Once the interview is closed it is possible for the Interview Manager program to check whether the record is clean or dirty. This is done by examining the FORMSTATUS of the record (see Figure 14). It is then possible to record that status in a field in the record for ease of access by users or other systems.

Figure 14. Segment of source code that checks the clean/dirty status

```

{check the instrument status and record the result in the BlaiseStatus field}
IF fInterview.FORMSTATUS = NOTCHECKED THEN
  fInterview.CHECKRULES
ENDIF
IF fInterview.FORMSTATUS = CLEAN THEN
  fInterview.BlaiseStatus:=BLClean
  aStatus := 'CLEAN' {Instrument status is CLEAN}
ELSEIF fInterview.FORMSTATUS = DIRTY THEN
  fInterview.BlaiseStatus:=BLDirty
  aStatus := 'DIRTY' {Hard errors exist in the instrument}
ELSEIF fInterview.FORMSTATUS = SUSPECT THEN
  fInterview.BlaiseStatus:=BLSuspect
  aStatus := 'SUSPECT' {Soft errors exist in the instrument}
ELSEIF fInterview.FORMSTATUS = NOTCHECKED THEN
  fInterview.BlaiseStatus:=BLNotchecked
  aStatus := 'NOT CHECKED' {Instrument status is NOT CHECKED}
ENDIF

```

9.3 Setting response status

Once the interview is closed the Interview Manager can intervene and provide the operator with a screen where the response status can be set.

The response status options are modified to match the completeness of the interview. The logic for completeness is best done within the Blaise instrument because then it is available to any system that needs this information. Completeness can be tested by placing particular points within the instrument flow then checking how many of those points have been triggered (see Figure 16).

Depending on the completeness of the interview, then only particular response status options are possible. The response status screen is set up so that only the relevant response settings are activated and others are not (see Figure 17). At this point there is also an option for the operator to re-enter the case record if desired. Remarks can also be added in the Response Status dialog and these will be inserted into the case record.

Figure 16. Segment of source code with typical logic for completeness of an interview

```
IF StartInterview=Continue THEN
  IF Endsurvey=Continue THEN
    IF Eligible=No THEN
      StatusFlds.Condition1:=3 (ineligible)
    ELSE
      StatusFlds.Condition1:=1 (full response)
    ENDIF
  ELSE
    StatusFlds.Condition1:=2 (partial response)
  ENDIF
ELSE
  StatusFlds.Condition1:=4 (empty record)
ENDIF
```

Figure 17. Screen capture of the response status dialog with relevant entries activated

Select Response Status

Codes available

- A. Complete interview
- B1. Partial interview, refused**
- B2. Partial interview, boarding**
- B3. Partial interview, other**
- C. Eligible, refused
- D. Eligible, refused entry
- E. Eligible, boarding
- F. Eligible, language
- G. Eligible, other
- H. Unknown, refused
- I. Unknown, boarding
- J. Unknown, language
- K. Unknown, other
- L. Missed, no interviewer available
- M. Missed, other
- N. Ineligible (Out of scope)
- O. Quota full
- P. Other

Response remarks:

Initial response status: B1

Re-enter eXit

Information
No remarks in this interview.

Upon completion of the Response Status dialog, control is returned to the Case Manager.

10. Summary

The programs described in this paper illustrate the kind of facilities that are needed to manage one or more surveys that involve the use of notebook computers in the field. It shows how there are three layers of management that need to be supported: survey management, case management and interview management.

For organisations that have little or no existing survey infrastructure, Maniplus clearly has sufficient functionality and versatility to provide the solution.

CAPI system at Central Statistical Bureau of Latvia

Karils Zeila, Norberts Talers, Pavels Onufrijevs, Central Statistical Bureau of Latvia

Objectives

The purpose of the CAPI system development is to increase the work quality and timeliness of the interviewer section in a sense that data collection can be made easier and faster for the interviewers. As the data is entered directly into a laptop no more data entry is necessary from paper forms, and the data validation procedures are carried out on – site, while interviewing the respondent. The gain of using mobile networks and GPRS is that interviewers are able to send the data virtually any time, for example after receiving the information from every respondent thus making the time from entering the information into the laptop to getting the data to the office very short and diminishing the problem of possible data loss due to some kind of a technical breakdown. Also, the management of interviewers work becomes easier and more straightforward while using CAPI technologies, because of possibility to control the process at any time.

At the beginning of 2004 a pilot project on data collection with 5 interviewers using laptops and Blaise was carried out; 100 respondents were questioned on Labour force survey. In 2005, on May, the development of CAPI system at CSB was started. One person was responsible for developing CAPI system, and two programmers were responsible for developing first questionnaire in Blaise. In December 2005 45 laptops were received, 45 interviewers were trained, and from the January 2006 the data in Labour force survey is being collected by using CAPI technologies. From middle of March 2006, EU – SILC survey data will be collected in this system also. CSB has plans to move all permanent surveys to CAPI environment by the end of 2007.

The system consists of four logical parts:

- Case management system on interviewers' laptops;
- Data transfer system via GPRS;
- Information management at the Office;
- SMS system.

Case management system

Case management system on interviewers laptops consists of two software types – for the data entry of questionnaires specialised data entry software Blaise is used; for the case management a tool developed by CSB programmers is used. When working with a laptop, an interviewer is working with the case management system with several possibilities:

- Receive the data on new surveys (respondent list and questionnaire itself)
- Send the data about surveyed respondents, and the data itself
- Work with received respondent list, surveying respondents

The core case management system is the place where an interviewer can see the respondent list for a survey chosen, with possibilities to set a meeting time for a specific respondent, or to open Blaise questionnaire, to which specific commands are passed to Blaise from case management system and start collecting data. Also, interviewers have possibilities to see the statistics on their work done (how many respondents have responded, how many non response, or how many not yet questioned). Different filtering functions are prepared to ease the work of an interviewer.

Whenever data sending and receiving is taking place, GPRS is used for that – on each of the laptops GPRS card is installed, which is used for connection to service provider. In case of receiving an update of the respondent list, or receiving completely new questionnaire with respondent list, interviewers just have to press one button, enter authorization information and wait for acknowledgement information from the system about successful data receipt, or about the failure. Technically, a script is activating GPRS connection on a laptop, connecting to an FTP server at office of CSB, and downloading a password protected zip file, from which upon the receipt

data is extracted and putted in correct place. Some file consistency checks are done in order to be sure that the file is not corrupted.

The gain of using mobile networks and GPRS is that interviewers are able to send the data virtually any time, for example after receiving the information from every respondent thus making the time from entering the information into the laptop to getting the data to the Office very short and diminishing the problem of possible data loss due to some kind of a technical breakdown. Also, the management of interviewers work becomes easier and more straightforward while using CAPI technologies, because of possibility to control the process at any time.

When the data is sent from the laptop to our office from point of view of an interviewers it is much the same as in case of receiving the information. Scripts are collecting the needed files to be transferred, zip them in a password protected archive and put on an FTP server at office of CSB.

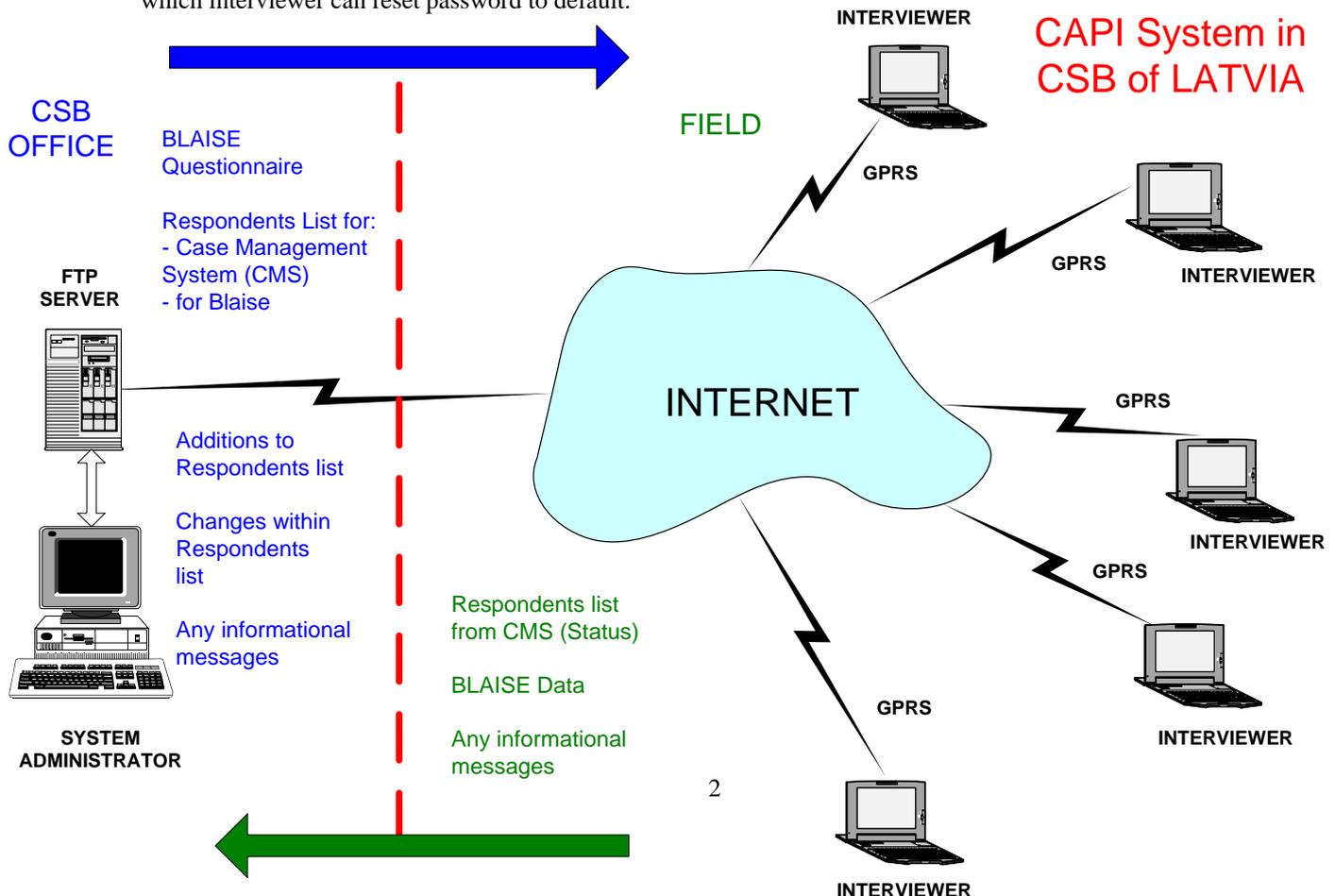
Information management at the office includes several parts:

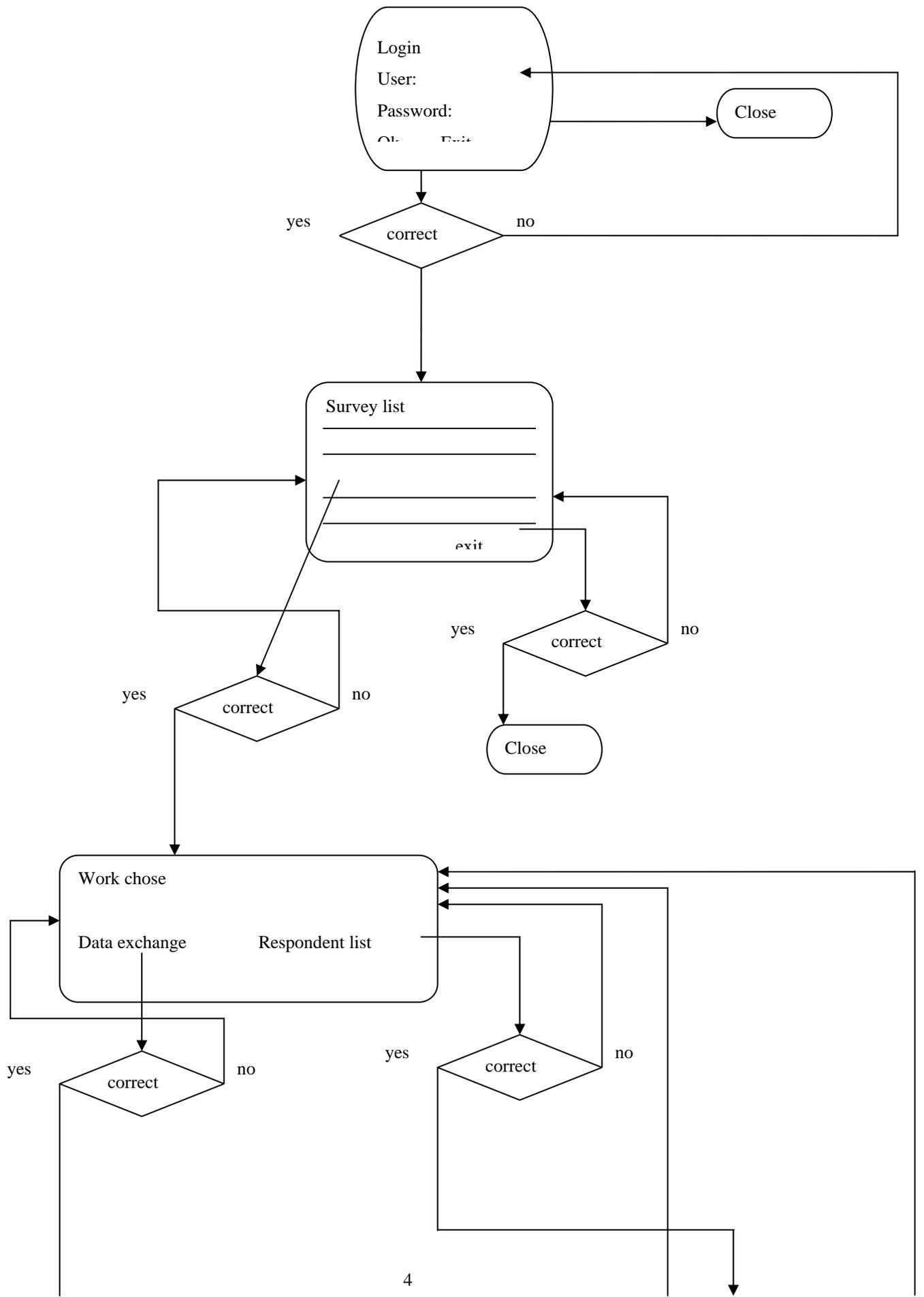
- A tool for respondent list separation in portions for each interviewer based on division by territories predefined.
- A tool for archiving of the respondent list part and the questionnaire, and putting that on an FTP server;
- Getting the information sent back by interviewers from FTP server, with possibilities for interviewers' section chiefs see the progress of work on surveying respondents.

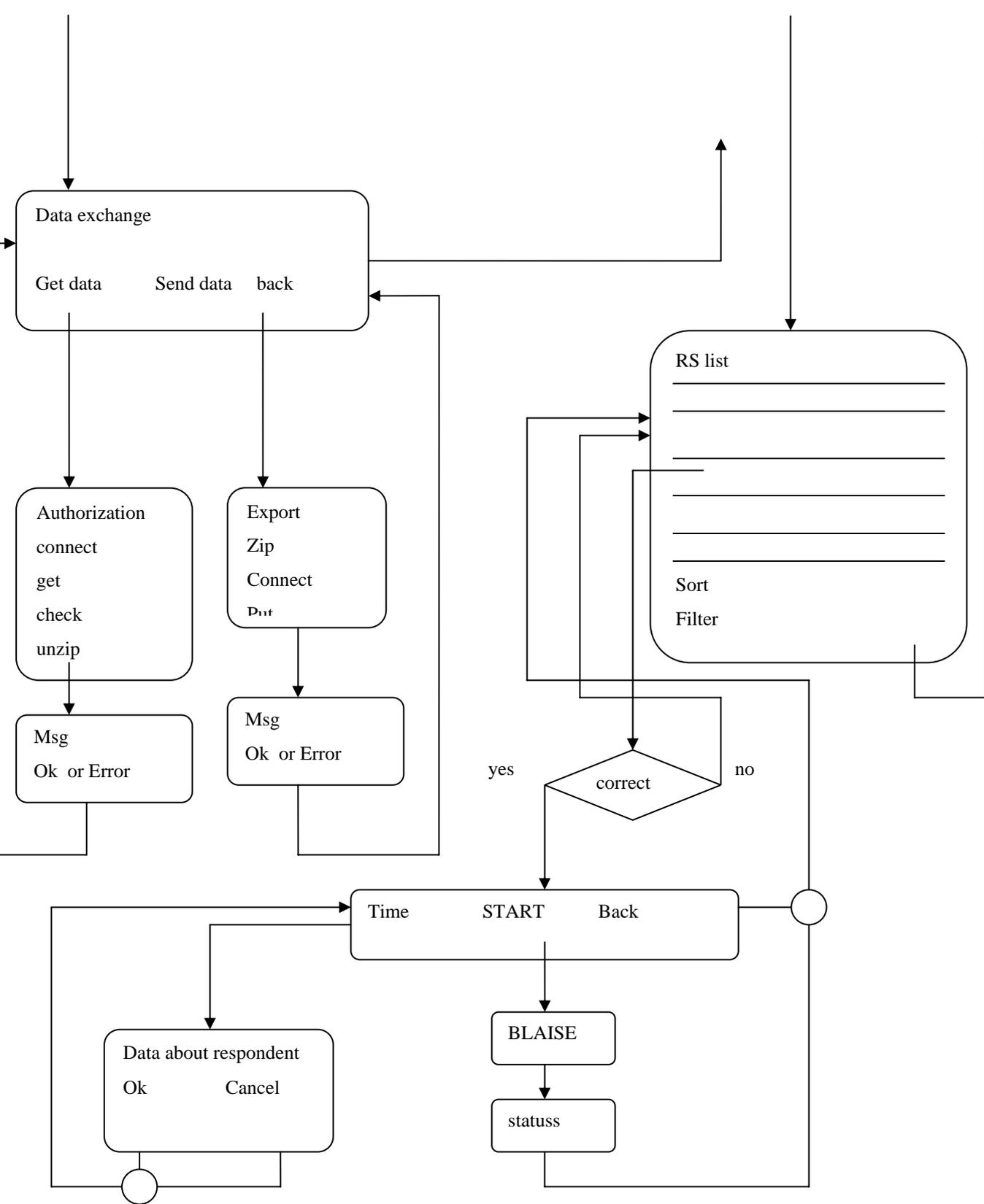
Sms system

SMS is short message service. In our case this is like short messages on mobile phones, with this kind of service we can send some instructions and *.doc *.xls files and receive some questions or comments from interviewers. Interviewers can send messages to each other or to CSB responsible person. System is standalone application on interviewer's computers, which use GPRS for data transmission. System was developed to reduce cost on phone calls to interviewers to instruct them. Stand alone solution was choised because, if one of systems brake down we still will have possibility to repair it through another one.

Some other tools and hidden features were developed during system run. For example if interviewer forgot his or her login and password for case management system, we can help them, with special tool keycode is generated on laptop and should be sent to CSB office, at office system administrator will generate the second keycode with which interviewer can reset password to default.







Minimizing the errors in the questionnaire and monitoring the survey process

Kimin Kim, Korea Labor Institute

1. Introduction

The Korea Labor Institute (KLI) first started CAPI (Computer Assisted Personal Interviewing) that is the face-to-face interview method by using computer in 2006 on the Korean Longitudinal Study of Ageing (KLoSA) and Workplace Panel Survey (WPS) for the purpose of improving data quality. It was also used for the Korean Labor & Income Panel Survey (KLIPS) in 2007, and today all three panel surveys use the method. Initial efforts were concentrated on ensuring smooth operation of the CAPI system, and it stabilized over the years through several surveys. KLI then started to study ways to improve data quality even further, and focused on the development of programs that could improve the accuracy of the survey and monitor the interview situation.

This paper looks into the two programs thus developed, in terms of how they are being used and the outcome. Chapter 2 reviews the program that allows the survey to be reviewed by displaying the entire content of the blaise datamodel file onto a paper format. Chapter 3 is about the program that analyzes the audit file that stores the entire computer-conducted survey process.

2. Questionnaire Review Program

2.1 Overview

CAPI is a face-to-face interviewing by using computer that replaces the paper questionnaire in a survey. The paper questionnaire must be completely and faithfully implemented onto the blaise datamodel program. The quality of the blaise datamodel program is essential because any errors will severely affect the data. But for such surveys of such complex structure and large size like the KLoSA, KLIPS and WPS, there is a high risk of errors when the questionnaire is implemented onto the blaise datamodel program. To minimize such errors, the questionnaires must be reviewed to the fullest extent possible by allocating much time and human resources.

KLI developed a questionnaire review program to enhance the quality of the blaise datamodel program in a quicker timeframe. How this program works is that it allows everything from the computer-implemented survey to be displayed onto a paper format, including the questionnaire logic, error checking and responses. It can even be downloaded as a .doc file to be printed out. This not only allows better scrutiny of the survey, but also a view of the overall survey flow by observing the responses. For example, the user can check whether the survey was conducted properly by reviewing the responses on some of the questions. This questionnaire review program later added the code book feature, which shows the frequency of a certain response for each question.

2.2 Questionnaire Review

Figure 1 is a screen shot of the questionnaire review program. The program can be reviewed by entering the BDB (Blaise DataBase) file to be searched in the "Database name" field. On right screen all variables can be viewed or different variables can be selected. In particular, certain questions can be categorized as one group, to always query the responses of those questions whenever viewing the same survey. The user can also display the survey according to specific query conditions (see <Table 1>). The response values can be viewed in all or selectively by the survey ID.

Table 1. Query conditions for the questionnaire review program

Query	Description
Questionnaire	To see the questions of the survey only, without the responses
Hard/soft	The Blaise program has "hard check" and "soft check" features. Hard check is for errors that should never happen, and soft check is for potential errors that need to be reviewed by the user. These two will be marked on the survey when this query condition is used.
Block name	When the survey is implemented on the Blaise program, certain chapters of the survey can be separated into blocks. This is the name for each block.
Box by group	To display each block in a separate box.
Responses Question	To see the responses.
Logic	To see the logic of the survey.

Figure 1. Survey Review Program

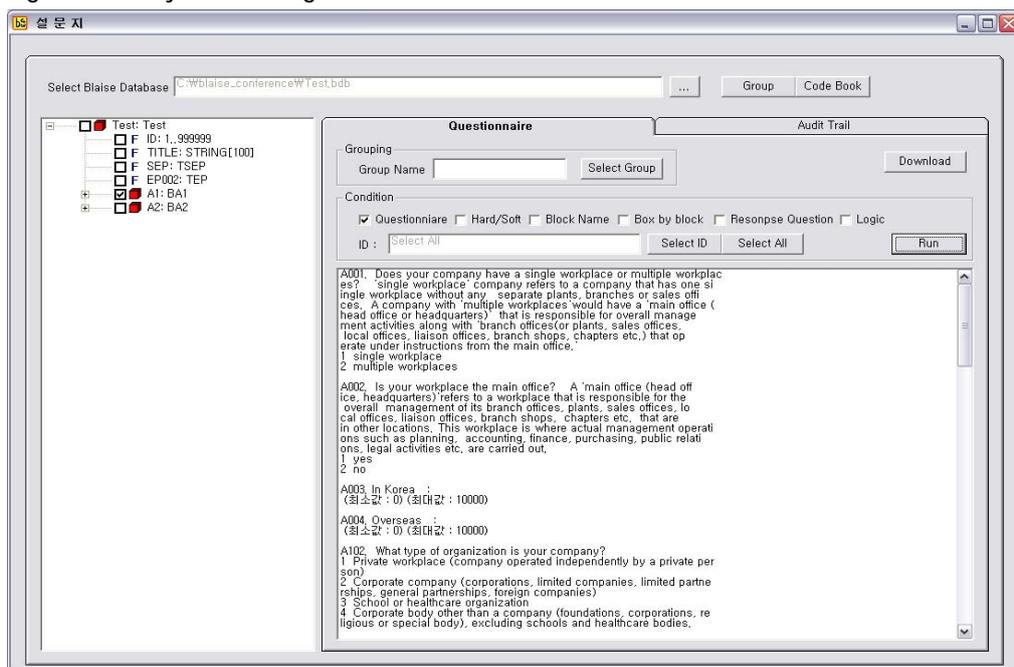
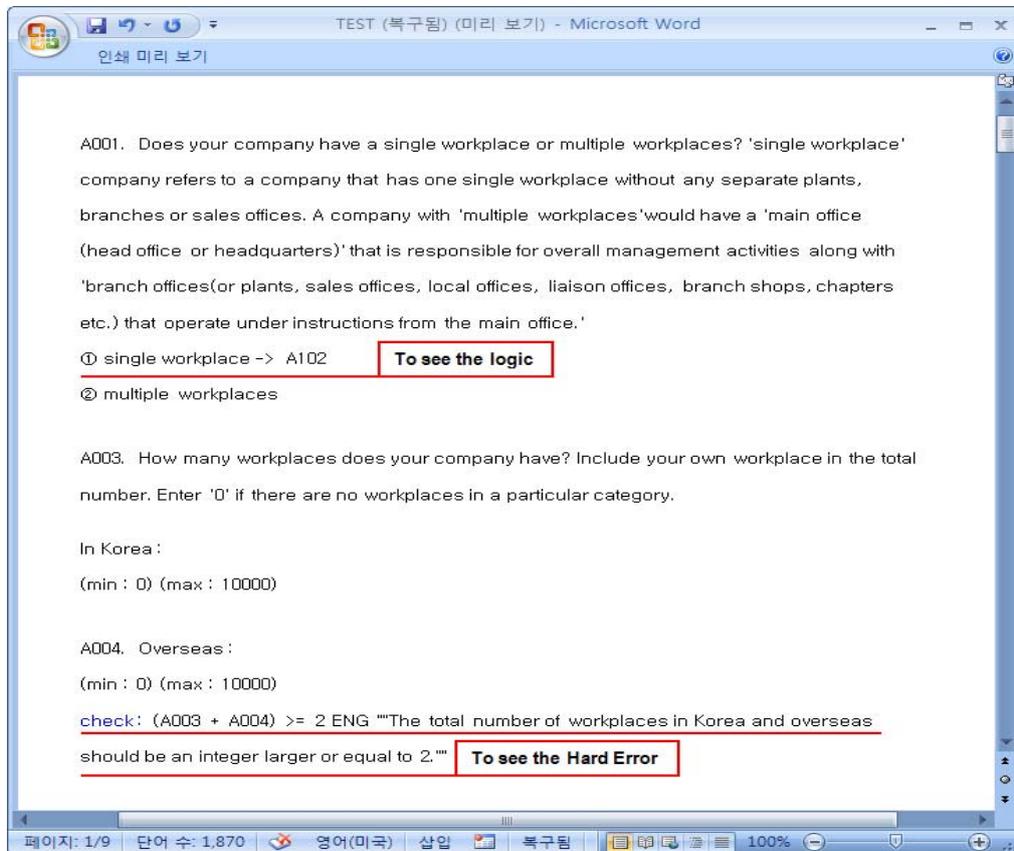


Figure 2 is the screen shot of the questionnaire review program downloaded as a .doc file. This is the result of a query using different variables, survey ID and other query conditions. The .doc file is edited for easier viewing than the computer screen.

Figure 2. Questionnaire in the MS Word Format



2.3 Code Book

The code book feature was added later on because the responses can be queried using the survey files. As can be seen in Figure 3, the code book is designed to show the survey questions and the frequency of the responses in an Excel format. We can know the distribution of answers by question because it is possible to show it

Figure 3. Code book

Fields	Question Text	Category	freq
A001	Does your company have a single workplace or multiple workplaces?	1 single workplace	1,060
		2 multiple workplaces	845
A002	Is your workplace the main office?	1 Yes	514
		2 No	331
A003	In Korea:		845
A004	Overseas:		845
A102	What type of organization is your company?	1 Private workplace (company operated independently by a private	71
		2 Corporate company (corporations, limited companies, limited	1,411
		3 School or healthcare organization	167
		4 Corporate body other than a company (foundations, corporations,	256
A103	Which of the following is most similar to the management system of your company?	1 An ownership management system where the owner has the authority	759
		2 An owner-centric management system where a professional manager	135
		3 A system where much management authority is transferred to a	433
		4 A professional managementsystem that is completely independent	316
		5 None of the above.	262

3. Audit File Analysis Program

Although data from a CAPI survey can be sent online immediately after the survey, it is still not possible to monitor all aspects of the survey in real time like CATI (Computer Assisted Telephone Interviewing),

particularly because wireless Internet is still not available anytime anywhere. Blaise makes up for it by allowing indirect monitoring of the survey using the Audit Trail feature. This feature records all movements of the keyboard or the mouse throughout CAPI onto a file named Audit. Figure 4 shows the Audit file format.

The purpose of this program is to indirectly assess the interview situation by analyzing the Audit file. Namely, it shows by survey ID, whether there were any out-of-the-ordinary trends, which questions took unusually long to respond, which questions required repeated corrections, among others. Identifying the difficulties in the survey and their causes will help improve the overall quality of the survey.

Figure 4. Audit file

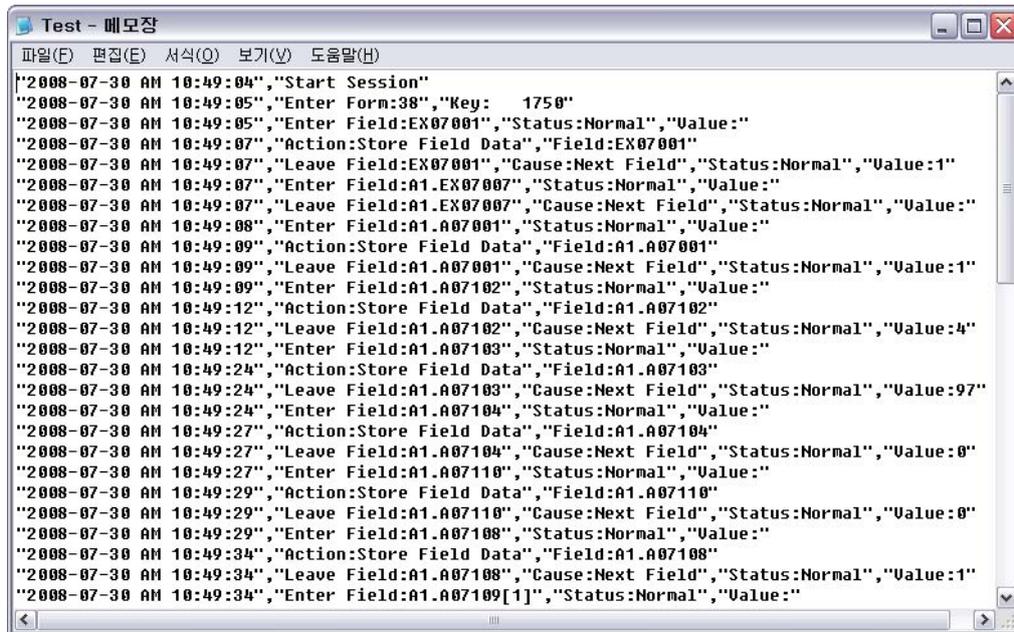


Figure 5 is a screen shot of the Audit file analysis program. For this program as well, the user enters the BDB (Blaise DataBase) to be viewed in the "Database name" field. Specific options can be entered on the screen on the right-hand side to view different time data. The required time data can be viewed in all or in part depending on the query condition (see Table 2). Also, This can be downloaded onto an Excel sheet to allow easy calculation of the time.

Figure 5. Log file analysis program

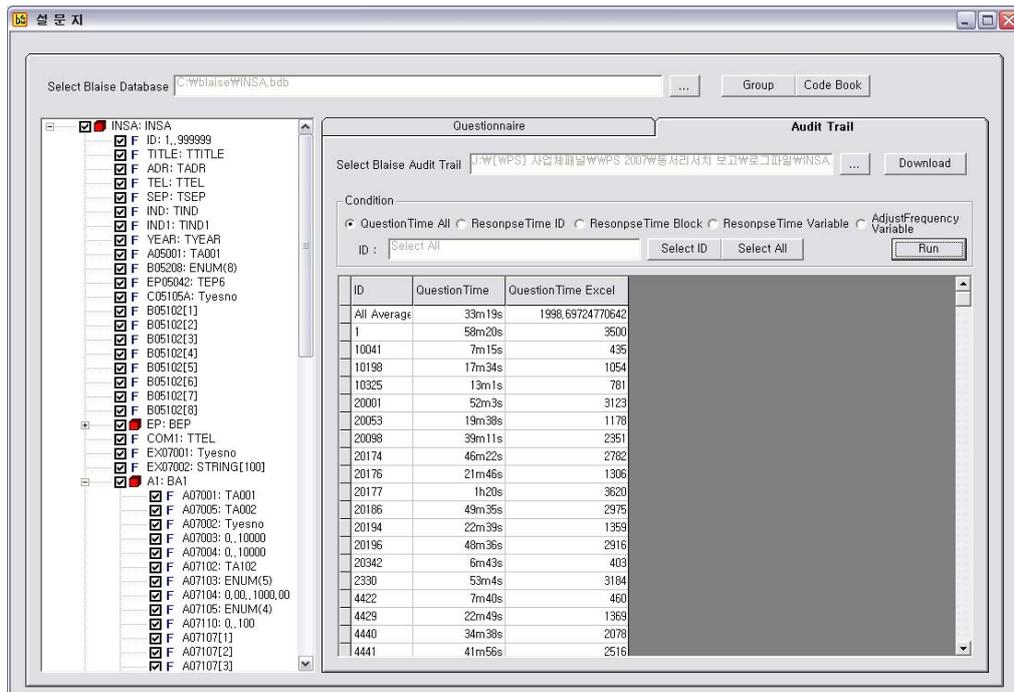


Table 2. Query conditions for the audit file analysis program

Query	Description
Question time all	To calculate between the start time and end time of the survey
Response time ID	To add all the response time for each variable
Response time block	The time required for each block (when the "block" feature is used for the survey program)
Response time variable	Response time by variable
Adjust Frequency Variable	Number of corrections by variable

4. Conclusion

KLI has been producing better quality data by adopting CAPI to conduct its surveys since 2006. As CAPI was smoothly implemented and stabilized, two new programs were developed in 2008 to review the survey tools and interview situation, as a way to enhance data quality even further.

One is a program that allows review of the survey by transferring the blaise datamodel program onto a paper form, to see the overall questionnaire structure and correctness of the response input. Initially, the purpose of this program was to enhance the accuracy of the survey. But not only that, but it also allows the user to see whether the survey was conducted appropriately, by showing the overall status of responses. In addition, the code book feature enables easy viewing of the data.

The other program is designed to enable indirect monitoring of the interview situation by analyzing the files that record the computer movements during the interview. The user can see how long the overall interview took, on which questions the respondents spent more time, how the responses for certain questions were corrected. This is likely to be useful for designing the surveys and training the interviewers. For the questions that took longer to respond, it can be discussed whether they need to be improved, and for those with many corrections, it can be determined whether they are so unclear as to confuse the respondents. Results of these discussions can be reflected to build better questionnaires and train the interviewers, to improve the overall quality of the survey.

For its CAPI, KLI adopted the globally-used Blaise program for the first time in Korea and successfully built a stable CAPI survey system. It continues to work toward improving its data quality, and to develop programs that will achieve this goal.

Macro-selection and micro-editing: a prototype

Wim Hacking, Statistics Netherlands

Introduction

Micro-editing procedures have a number of problems: often there are many checks with narrow tolerances resulting in too many mistakes that need to be resolved manually by analysts. The analysts cannot assess the relative importance of these errors. Each marked item has the same weight and needs the same amount of time for correction. However, many errors have a negligible impact on the final estimates: either they are small or they cancel out.

In general the boundaries for micro-checks are set conservatively, i.e. only error-less records are accepted. An example of a frequently used edit is that the relative change compared to the last period cannot exceed $\pm 10\%$. Generally, this approach results in a considerable amount of over-editing.

To address the issue of over-editing one can use macro-editing; this method is already in use at a number of places at Statistics Netherlands and other statistical offices, based on applications tailored for (a number) of statistics. An important difference compared to micro-editing is its use of aggregates, to assess the relative importance of a set of records. This is done by comparing concept-publications with those of the last period, **as a final check**. The largest deviations are either displayed on top or marked visually (e.g. by using colours). Based on suspect publication data the underlying data can be viewed, checked for inconsistencies and corrected; immediately after the correction at the micro-level one can check whether the reviewed publication data fits better in the time series. Another example of macro-editing is outlier detection where the (multivariate) distribution determines which records are suspect and possibly need to be corrected. Such a distribution can be obtained by comparing with $t-1$ data, but also by considering two (or more) related variables in one (or multiple) scatter plots; in these plots outliers may be marked based on visual inspection. Alternatively, these outliers may be determined automatically and subsequently be displayed, e.g. by colouring these points inside a scatter plot. A third example is prioritising (based on measures supplied by the analyst) of records: records having the highest plausibility index are listed first. This allows more direct and efficient manual editing. Some simulation studies (Granquist) report an efficiency gain between 35% and 80%.

These macro-editing approaches not only result in less work for the analyst, but also in a more meaningful analysis and editing process. He can always see the implications of his corrections at the micro-level on the output-aggregates.

These macro-editing techniques can also be used for the evaluation of (incoming) register data: based on appropriate aggregates one can compare the current version of the register with previous ones at the macro-level. After finding differences at the macro-level one can zoom in on more detailed levels and, finally, inspect and correct records.

Although applications of the principles above exist at various statistical offices but no general software has been made yet, which can be applied to different statistics, to our knowledge. During the last half year a prototype has been developed at Statistics Netherlands that has been designed to serve as such a tool. This doesn't mean that the tool is finished; in a certain way it's just a starting point to discover which macro-detection strategies work in practice and which not. This discovery process is still going on. Currently, we are working with data from road-transport and production.

Main idea

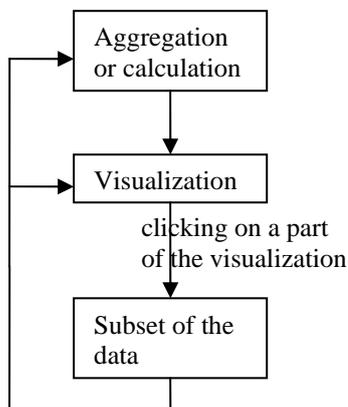
Macro-selection is always based on a collection of records to make a selection of those records that contribute the most to errors observed at some aggregate level. In order to assess this error one must have a reference set: this may be the data set itself (e.g. using outlier detection) or (aggregated) reference data (using $t-x$ data or data strongly correlated to the variables under study). The reference datasets can be used to make a prediction D_p of the current data D_t and the relative differences between the prediction and the actual values can be used to look

for possible errors. For this, the program has to be able to describe and access reference data, either micro-data or aggregated data.

During the macro-selection process, the data needs to be displayed in various ways to enable the analyst to zoom in or analyze the data. Looking at outliers and deviations at the aggregate level the analyst can either zoom in or apply another visualization, e.g. other variables in a scatter plot. Finally, at the most detailed level, the analyst needs to be able to edit a single record. To configure all of these possibilities, a project file can be specified in which the analyst needs to specify the elements of the editing process and, finally, the interactive process itself:

- 1 The input of the editing process: a data model of the microdata input and of the reference (aggregate) data.
- 2 The variables and their derivation in the aggregate(s) (e.g. plausibility functions, distribution moments, ...)
- 3 Visualizations: what data needs to be displayed, which colours needs to be used, etc.
- 4 The behaviour of the macro-edit program when interacting with the analyst. For example, when the analyst selects a cell from an aggregate: should a sub-aggregate be displayed or a scatter plot?

Based on the elements defined above the selection process can be depicted as follows:

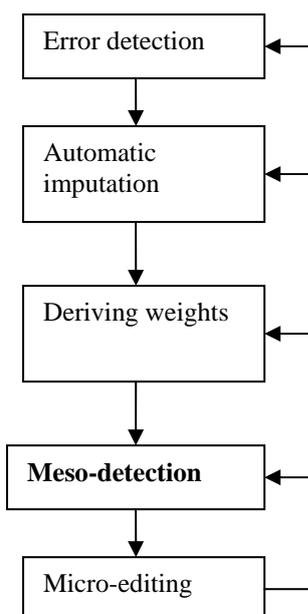


To make this flexible one needs a flexible description of the data that is being transported from one visualization to another. For example, consider an aggregate by A with possible values $(1, 2, 3)$ and variables $AVG(X)$, $AVG(Y)$ and $AVG(Z)$; the selected cells $A=1$ are selected from this aggregate can be considered as:

- As one (aggregated) record : $(1, AVG(X), AVG(Y), AVG(Z))$
- As a selection on the original microdata: all microdata records having $A = 1$

It means that the information passed on between visualizations must contain information on both aspects.

There are also some additional, practical, requirements: the macro-editing process often requires many records, e.g. millions of records. Even in the presence of such large amounts of records the program should still maintain its interactive character, i.e. the response-time (especially when records have been edited and tables or plots need to be updated) must be small. Secondly, the program must have an open character: the current version already uses some external programs to do its task (Blaise for editing, Bascula for (re)weighting and R for things like outlier detection). At the same time, to the analyst it should appear as if he's working with one tool. In other words, the integration between the macro-edit tool and the external programs should be quite good. Last, but not least, the tool needs to be flexible, so that it can be used for all kinds of statistics; in the current prototype a text-based user-interface is used to enable this. In a later version, a graphical user-interface will be added.



In the rest of the paper we will describe the elements of the specification in more detail and show some results for **production** data. The specifications are defined in a Blaise/Manipula-like syntax and in each section an example specification is shown and described.

Specifying the input

Microdata is specified as a Blaise datamodel, albeit with a few additional properties per field.

```

DATAMODEL MyData
PRIMARY
  ID
FIELDS
  RitAfst:real
  RitAant:integer
  rittype: integer
  btogew: real, ifmissing(0.0)
RULES
  rittype < ritaant
ENDBLOCK
  
```

When a value is missing, one can specify what default value should be used instead. In this example, `ifmissing(0.0)` means that whenever the variable `btogew` is missing, it is being replaced by 0.0.

In addition to the (current) microdata, one needs to specify the aggregate data. For this, two sections are added: `AGGREATEBY` and `FILTERBY`:

```

DATAMODEL MyAggData
AGGREGATEBY
  AggDef = Region * CompanySize
  AggDef2 = Region
  AggDef3 = CompanySize
FILTERBY
  Filter1 = "Profit > 1000"
FIELDS
  Region: string
  CompanySize: string
  avg_distance_now:real
  avg_distance_prev:real
ENDMODEL
  
```

These aggregate data models have two purposes:

1. to describe *existing* aggregate data (e.g. *t-1* data or any other correlated data); in this case one needs to specify what variables have been used to aggregate (AGGREGATEBY); also a previous selection (before the aggregation) can be specified (FILTERBY). Precisely one aggregate definition must and at most one filter can be specified.
2. to describe *new* aggregates: in this more than one aggregate and multiple filters can be specified and be combined.

Specifying the aggregation

Based on the datamodels above, one can define various aggregate functions to detect discrepancies between current and reference data. These functions can be

- a. Distribution properties of the microdata, e.g. its variance.
- b. Processing properties, such as percentage of item non-response or percentage of previously automatically imputed values in field X
- c. Plausibility functions: e.g. the relative change between weighted t-1 and t data:

$$\Delta = \frac{\left| \sum w_{i,t-1} V_{i,t-1} - \sum w_{i,t} V_{i,t} \right|}{w_{i,t-1} V_{i,t-1}}$$

where $w_{i,t(-1)}$ are the weights and $V_{i,t(-1)}$ are the values of one of the variables.

An example of each is contained in the following example section (5a, 5b and 5c)

```

AGGREGATE MyAgg                                     {1}
INPUT
  CurrYear : MyData                                  {2}
  LastYearAgg : MyAggData                            {3}
OUTPUT
  outputagg : MyAggDataWithDifferences                {4}
CELLS
  sum_distance_now := SUM(CurrYear.distance);        {5}
  sum_distance_last := LastYearAgg.sum_distance;
  z := VAR(CurrYear.Profit);                          {5a}
  PercFilled := COUNT(CurrYear.distance)/COUNT(ALL); {5b}
  difference := ABS(LastYearAgg.sum_distance - AVG(CurrYear.distance *
CurrYear.Weight))/ LastYearAgg.sum_distance {< 0.1}; {5c}
CELLCOMPARE
AggDef2:                                             {6}
  test1 := avg_distance_now('west')
           /avg_distance_now('east') > 2.0
           "There should be more transport in the west of the
           Netherlands";

ENDAGGREGATE

```

In the text above the following things are specified:

1. We define an aggregate named “MyAggData”

- Each aggregate has *one* microdata input which contains the current data that needs macro-editing. CurrYear is an internal identifier which can be referenced from the cells section, e.g. CurrYear.Distance. The reference MyData refers to datamodel defined earlier on.
- Each AGGREGATE has one or more additional reference aggregates, that can be joined to the aggregated microdata (i.e. CurrYear). An example of such an aggregate may be *t-I* data.
- The output refers to the datamodel of the aggregate as calculated in this AGGREGATE section.
- The section CELLS contains the derivation of the output variables. These can be simple aggregates (like SUM(X)), but can also contains a mix of variables form the various input sources. Note however, that the variables from the microdata can only enter these expressions inside an aggregate function, e.g. SUM(X). Available aggregates are AVG(X), STDEV(X), MIN(X), MAX(X), COUNT(X), SUM(X), MEDIAN(X), ALPHATRIMMEDMEAN(X, AlphaLEFT, AlphaRight, UseMedian), INTERQUARTILE(X)
- Finally, an additional set of **ratios** can be derived based on the aggregate data. For example **ratio test1** expresses the check that generally "There should be more transport in the west of the Netherlands". The expression avg_distance_now('west') indicates the column avg_distance for the aggregate Region='west'.

In addition, it's possible to use conditions inside an aggregate function like SUM, AVG, etc.:

```
TotalDistance := SUM(
    IF RitInd = 'R' THEN
        weight * Dist
    ELSEIF RitInd = 'Z' THEN
        weight * Dist
    ELSE
        0.0
    ENDIF
);
```

Specifying the visualizations

In each visualization the analyst can click on certain parts, leading to a sub-selection of the data under study. For example, by clicking on a cell from an aggregate (aggregated by A en B, say), all (micro)records from that cell (from A=1 en B=2, say) are selected. The prototype currently contains three kinds of visualizations:

- Report: at the end of each aggregate a number of **ratios** can be defined. These numbers from each calculated aggregate are shown in a overview report. Clicking on a **ratio** can be used to show the underlying aggregate.

Report

DASHBOARD

Aggregate	Test	Description	Value
av	test1	there should less wages for GK = 1	1
av	test2	item-nonresponse should not exceed 10%	0
av2	test3	there should less wages for SBI = 1	0

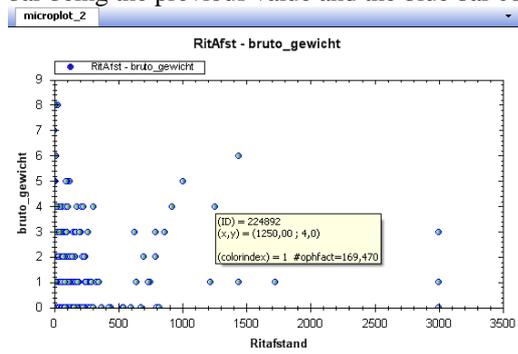
In this report three ratios are shown for two aggregate (*av* and *av2*). Two of the ratios are not OK and are shown in red.

- (Pivot) tables: used to show either aggregated or micro data. It's possible to give cells from variable V_1 a background colour based on a variable V_2 .

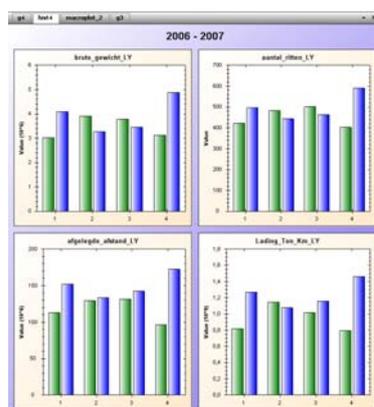
g2	macroplot_1	MicroPlot_1	g1			
	Nstrcode_char1	1	2	3	4	
	---	---	---	---	---	
	0	3068522	2961753	3273689	4082655	
	1	6188031	7643291	5904980	4823722	
	2	18114	384	54567	11750	
	3	1243028	1180896	1283409	2542215	
	4	596269	524567	459491	1167868	
	5	626735	756064	761129	826203	
	6	6936118	8174316	8087157	9612530	
	7	956003	1571037	821416	766716	
	8	4531556	4999253	4471983	7338516	
	9	8892334	8592738	8365994	11424068	

This pivot table shows $SUM(distance)$ as a function of the two aggregation variables `Nstrcode_char1` (0...9) and `Quarter` (1...4). The color is based on the relative difference with the corresponding aggregated value from last year, i.e. 0% is mapped on green and 100% (or more) is mapped onto red.

3. Plots: currently there are two kinds of *interactive* plots
 - a. Scatterplots (shown in (a)): the colour of the plotted points may be coloured based on a third variable and a tool tip text may be defined that shows additional info when hovering over a point.
 - b. Barplots (shown in (b)) : in this specific plot 4 variables are show per quarter, with the green bar being the previous value and the blue bar corresponding to the current value.



(a)



(b)

Specifying other parts

Editing macro-data

Essential to macro-diting is a close integration with a microdata editor. In this prototype the Blaise editor (DEP) is used. To edit a user-selected record it is converted from database table(s) into a Blaise record; subsequently, the Blaise editor is started. After the editing of the record has finished the Blaise record is converted into the database table(s). The altered fields with old+new values are passed back to Macroview so that it can adjust existing aggregates and visualizations in an efficient way.

Using other external software

One of the techniques for macro-detection is the use of outlier-detection. This may require many numerical algorithms which have already been implemented in a number of (open-source) packages. The same holds for the calculation of weights (Bascula, among others). As an example MacroView can specify an interaction with the package R (see references); this is a widely used open-source package allowing matrix-based numerical and statistical calculations. Many additional R packages have been written by the academic community, based on the R scripting language.

Specifying interaction

Now that the elements have been specified, the interaction during the analysis can be described:

Process

```
a(PrevData, PrevDataMacro)
a.Done -> av(CurrData, PrevDataMacro, CurrDataMacro)
av.Done -> grid1(Agg.OutputData) AT Test Position Bottom

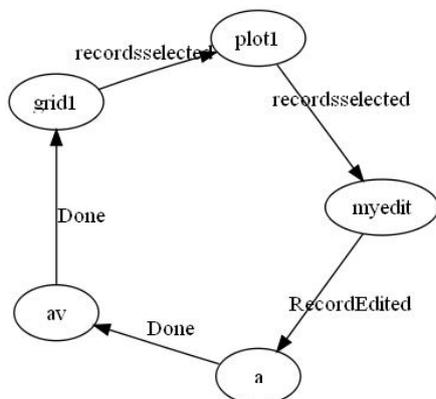
grid1.recordsselected -> plot1(grid1.outputdata) at Test Position Top
plot1.recordsselected -> MyEdit(plot1.outputdata)
MyEdit.RecordEdited -> av(CurrData, PrevDataMacro, CurrDataMacro)
```

Endprocess

For clarity, the process has been depicted below. The arrows correspond to so called events. In this example there are two events:

- Done: the aggregate has been calculated
- RecordsSelected: the user selected either a cell from the table or a point from the scatterplot.

Note that the process forms a loop as shown below: the effects of the edit cause recalculation of aggregates and result in altered tables and plots being displayed.



Discussion

This paper presents a first prototype attempting to generalize the concept of macro-analysis. Currently only a limited number of visualizations have been implemented. Also, the prototype uses a text-based user-interface, as shown in several sections in this paper; this is meant for prototyping purposes and more advanced users. For less experienced or incidental users a graphical user-interface will be added to aid in the specification of the setup for the macro-editing session. Another addition might be a so called wizard dialog to help the specification of a new setup.

One of the requirements for the macro-edit tool is its interactive character: any of the steps, especially the edit step, shouldn't take too long, i.e. no longer than 2 or 3 seconds under normal circumstances.

However, this may not always be possible: the plausibility functions might be complex or an external program to re-apply imputations or weighting might take some time.

At the moment we're testing the prototype both on economical data (production statistics) and road transport. Trying to serve both statistics will help in generalizing the concepts needed for macro-editing. This may result in more flexible ways to specify plausibility functions, other visualizations or more interactive options when analyzing the data. This iterative process will have the main focus in the near future.

References

Granquist, L. (1994). Macro editing: A Review of some Methods for Rationalizing the Editing of Survey Data. Statistical Data Editing, Volume No. 1: Methods and Techniques.

De Waal, T. and Haziza, D. (2008), Statistical editing and imputation. Handbook of Statistics, Volume 29, Sample Surveys: Theory Methods and Inference, Editors: C.R. Rao and D. Pfeffermann.

The R project for statistical computing: see <http://www.r-project.org>

The Three-Legged Stool Is Now a Four-Legged Chair: Specification Guidelines for Blaise Survey Instruments

James R. Hagerman, Jody L. Dougherty, and Sue Ellen Hansen The University of Michigan

1. Introduction

Much has been written and presented in past International Blaise User Conferences about the integration and use of good screen design, programming guidelines, and consideration of data analysis and documentation requirements; the three-legged development stool (Couper 2000). Believing that instrument design and cost effective Blaise programming is facilitated by good instrument specifications. The University of Michigan's Survey Research Center (SRC) has put a considerable amount of effort into developing standards for creating specifications for Blaise survey instruments.

Between 1999, when SRC started programming instruments using Blaise, and completion of the guidelines in 2007, there was wide variation in the quality of specifications provided to SRC programmers, which generally reflected each survey manager's style, experience, and time available to develop specifications. There were some general rules of thumb that were followed; but, in many ways, it was like the Wild West when it came to how specifications for Blaise survey instruments were written. Adherence to screen design standards and programming standards was marginal at best.

In 2007, a proposal was written and funded to develop and publish a book of standards for use in the Survey Research Center that encompassed all of the items listed above; screen design, programming, data documentation, and finally writing specifications for Blaise survey instruments.

This paper will discuss the development of that publication and the specifications chapter in particular. Besides discussing the process and content of the book, samples of it will be included as well as a discussion on the desire to design an experiment which would measure and confirm the accumulating anecdotal evidence of decreased Blaise programming costs and errors since the publication of the book.

2. Early Days of Specifying for Blaise Survey Instruments

In SRC experience with programming Blaise instruments, the quality and cost of Blaise programming has been directly affected by the length of the development period, the hours budgeted for programming, and the quality of the instrument specifications. In the early days of SRC Blaise programming, programmers started with either a legacy questionnaire, many times the old "box and arrow" questionnaire itself, or specifications in the form of a questionnaire produced by the survey manager, in a format she was comfortable producing, often with little understanding of the requirements of computer-assisted survey instrumentation or Blaise.

In the absence of a coherent set of guidelines for specifying Blaise survey instruments, many survey managers were left to their own devices in terms of what they prepared and delivered to the CAI programmer to be developed into a Blaise survey instrument. This practice, depending on the survey manager and CAI programmer involved, would often lead to inconsistent design, poor communication, many programming iterations, a more arduous testing period, and higher costs to the project in terms of programmer, tester, and the survey manager time. In addition, there was often inconsistency in screen design across instruments, which affected users (interviewers). Projects programmed by different CAI programmers would literally have a different look and feel. This result could hinder the interviewer's efficiency because of the subtle but significant differences in the screens and behavior of the application. Below, are some examples of specifications used prior to the publication of the SRC Blaise Standards.

2.1 Specification Examples Pre-Standards

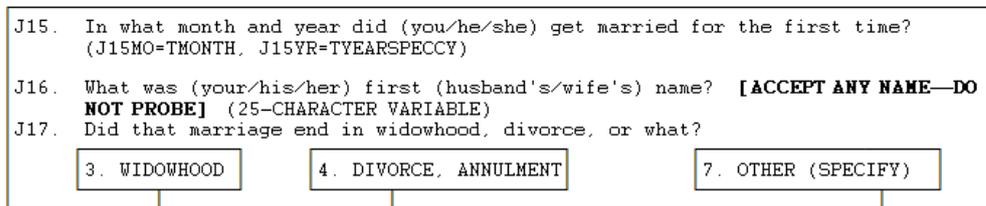
Here are two early examples of specifications used to program Blaise applications. This type of specification closely resembles a “box and arrow” survey instrument.

2.1.1 Example A

J12a. AQSN OF THIS PERSON
 J13. INTERVIEWER CHECKPOINT (VARNAME=J13CKPT)



J14. [INCLUDE THIS SENTENCE ONLY FOR VERY FIRST ITERATION THROUGH SECTION J: Now I'd like to ask about (your/his/her) family history.] Altogether, how many times (have you/has [he/she]) been married? (TWO DIGITS)



J18. In what month and year (were you/was [he/she]) widowed? (J18MO=TMONTH; J18YR=TYEARSPECCY)

J19. In what month and year did (your/his/her) (divorce/annulment) become final? (J19MO=TMONTH, J19YR=TYEARSPECCY)

J20. In what month and year did (you/they) stop living together? (J20MO=TMONTH, J20YR=TYEARSPECCY)

J21. In what month and year did (you/he/she) (last) get married? (J21MO=TMONTH, J21YR=TYEARSPECCY)
 J22. What was (your [husband's/wife's]/his wife's/her husband's) name? **[ACCEPT ANY NAME—DO NOT PROBE.]** (25-CHARACTER VARIABLE)
 J23. INTERVIEWER CHECKPOINT (VARNAME=J23CKPT)



J24. In what month and year (were you/was [he/she]) widowed? (J24MO=TMONTH; J24YR=TYEARSPECCY)

J25. In what month and year did (your/his/her) (divorce/annulment) become final? (J25MO=TMONTH, J25YR=TYEARSPECCY)

GO TO KIDS

J26. In what month and year did (you/they) stop living together? (J26MO=TMONTH, J26YR=TYEARSPECCY)

2.1.2 Example B

R1. Now I would like to ask you a few questions about help you might have received two years ago. At any time during 2001, even for one month, did (you/anyone in this family) receive any public assistance or welfare payments from the state or local welfare office? [IWER: DO NOT INCLUDE FEDERAL FOOD STAMPS OR SSI. DO INCLUDE ADC, AFDC/TANF, GENERAL ASSISTANCE PROGRAMS, EMERGENCY ASSISTANCE, CUBAN/HAITIAN REFUGEE, OR INDIAN ASSISTANCE.]

1. YES

5. NO ———GO TO R7

R2. Who received that public assistance or state or local welfare in 2001?
(TWO-DIGIT MULTIMENTION: AQSN OF EACH SUCH PERSON. IF DK/RF, GO TO R7)

R3. In which state was (PERSON IN R2) living at the time (he/she) received that public assistance? (TWO DIGITS: R3STATE USING TSTATE)

R4. Which type of public assistance did (you/PERSON IN R2) receive? (Any other?)
[ENTER ALL THAT APPLY]

1. (STATE-SPECIFIC ASSISTANCE-- SEE LIST)

2. ADC/ AFDC

3. GENERAL ASSISTANCE

4. EMERGENCY ASSISTANCE

5. CUB./HAIT. REFUGEE

6. INDIAN ASSISTANCE

7. OTHER (SPECIFY)

ASK R3-R4 FOR EACH PERSON MENTIONED AT R2

R5. How much did (you/everyone in your family) receive altogether from all of the public assistance or welfare programs you just mentioned during 2001? (TWO VARIABLES: R5YRAMT AND R5MOAMT, EACH 7 DIGITS)

R6. During which months did (he/she) receive any type of public assistance or welfare during 2001? (MULTIMENTION: R6x, USING TMOSTRING)

R7. At any time during 2001 did you (or anyone else in your family living there) receive Supplemental Security Income?

1. YES

5. NO ———GO TO R11

R8. Who (in the family) received Supplemental Security Income?
(TWO-DIGIT MULTIMENTION: AQSN OF EACH SUCH PERSON.)

R9. How much did (you/he/she) receive altogether from Supplemental Security Income during 2001? (TWO VARIABLES: R9YRAMT AND R9MOAMT, EACH 7 DIGITS)

R10. During which months of 2001 did (you/he/she) receive it?
(MULTIMENTION: R10x, USING TMOSTRING)

ASK R9-R10 FOR EACH PERSON MENTIONED AT R8

The next two examples of specifications begin to show signs of the progression towards what would become the standardized format. The improvements included listing out each question in a top-down style instead of questions all over the page with routing implied by boxes and lines or arrows. Each question also possessed its own coding format and routing instructions.

2.1.3 Example C

V10. Altogether, how many nights were you a patient in the hospital in the last year?

_____ (Number of times)

998 = Don't know

999 = Refused

➤ VALID CODES: 1-365

➤ "SOFT" CONSISTENCY CHECK: IF IWER ENTERS NUMBER GREATER THAN 60, ASKIWER TO VERIFY THE ENTRY.

➤ ANSWER TO V10 MUST NOT EXCEED ANSWER TO V7.

V11. Were the costs for your hospital stay(s) completely covered by Medicare, Medicaid, or other health insurance, partly covered by insurance, or not covered at all by insurance?

1 = Fully covered

3 = Partly covered (or covered with a copay)

5 = Not covered at all

7 = [IF VOLUNTEERED:] Costs not settled yet

8 = Don't know

9 = Refused

V12. In the last two years, have you been a patient overnight in a nursing home, convalescent home, or other long-term health care facility?

1 = Yes (Go to 13)

5 = No (Go to 19)

8 = Don't know (Go to 19)

9 = Refused (Go to 19)

2.1.4 Example D

GLOBAL PROGRAMMING ATTRIBUTES:

Allow global DK and RF on all fields unless otherwise specified

Do not allow EMPTY on any fields

Consistency checks only where specified

Standard SRO formatting guidelines on all screens

VOL STMT

Voluntary Statement

[^Display Respondent Name](#)

A few months ago you participated in a study of social relations. We would like to thank you once more. We are calling you with some follow up questions. This should only take about 10 minutes.

As in the previous interview, your participation is completely voluntary and confidential. If we should come to any question you do not want to answer, let me know and we'll go on to the next question.

[Enter \[1\] to continue](#)

Question Type: Numeric; Range 1-1

A3.

People do different things to help them cope with life events. We'd like to know what you did to cope with the event or problem you experienced. Did you do any of the following?

- A3a. take any specific action to try to solve the problem
- A3b. do something to help you relax
- A3c. distract yourself by thinking about other things or engaging in some activity
- A3d. try to see the event in a different light that made it seem more bearable
- A3e. express emotions to reduce your anxiety, frustration, or tension about the event
- A3f. seek spiritual support or comfort concerning the event
- A3g. seek emotional support from loved ones, friends, or professionals concerning the event

CORE_NETWORK

A4

Which two people helped you the most during this event?

PROBE FOR TWO MENTIONS. IF EITHER PERSON NAMED IS A PROFESSIONAL SERVICE PROVIDER (e.g. DOCTOR, PHARMACIST, SPIRITUAL LEADER) ASK R TO NAME SOMEONE WHO IS NOT A PROFESSIONAL SERVICE PROVIDER)

PERSON 1 string 25

PERSON 2 string 25 (allow empty if no second mention)

**programmer - we want to loop through the A5-14 for each person mentioned at A4.*

2.2 The Development of SRC Blaise Standards

Prior to initiating the project in 2007 to create a set of standards for screen design, programming, documentation, and specification, a variety of documents and guidelines existed in many places and were of varying age and quality.

In an effort to consolidate and update all of these documents, the SRC Blaise Standards were conceived. For the first time, an importance was assigned to making these standards a cogent and useful tool for programmers, specification writers, survey managers, researchers, and testers.

It was believed that by standardizing the look and format of instrument specifications, they could become a single resource for a wide range of users throughout instrument development, data collection, and data documentation. Thus, the new standardized specifications became a useful document and communication tool for all phases of the survey lifecycle.

Following are short examples of delivered specifications that were written to conform to the SRC Blaise Standards for specifying a Blaise survey instrument. Section 2.2.1 shows a portion of the original specification delivered for programming. The same portion of specifications as re-written to conform to the new standard is displayed in section 2.2.2. Section 2.2.3 shows another example of specifications based on the new standards.

2.2.1 Original Non-Standard Specification

Section 12: Self Beliefs

The following is a list of statements that you may or may not identify with. Read each statement carefully. Then, using the scale shown, please rate the extent to which you identify with the statement and select that box.

Response scale: 1 = not at all like me -- 7 = very much like me

- | |
|---|
| 1. If I ruled the world it would be a better place. |
| 2. I can usually talk my way out of anything. |

Section 13: Personal Thoughts and Behaviors

Using the scale below as a guide, write in a number from 1 to 5 to indicate how much you agree with each of the following statements.

1	2	3	4	5
Disagree Strongly	Disagree	Neutral	Agree	Agree Strongly

- ___1) **Not** hurting others' feelings is important to me.
- ___2) I think I could "beat" a lie detector.
- ___3) I'm a rebellious person.

2.2.2 Specification Using New Standard

=====

Section 12: Self Beliefs (SB)

=====

SBIntro

The following is a list of statements that you may or may not identify with. Read each statement carefully. Then, using the scale shown, please rate the extent to which each statement describes you.

/"Self Beliefs Intro"

◆ Press [ENTER] to continue

=====

SB1

If I ruled the world it would be a better place.

/"World better place"

Not at all	1
A little	2
Somewhat	3
Quite a bit	4
Very much	5

=====

SB2

I can usually talk my way out of anything.

/"Smooth talker"

Not at all	1
A little	2
Somewhat	3

Quite a bit 4
Very much 5

=====

=====
Section 13: Personal Thoughts and Behaviors (PTB)
 =====

PTB1

Using the scale listed as a guide, select the number from 1 to 5 to indicate how much you agree with each of the following statements.

Not hurting others' feelings is important to me.

/"Not hurt others' feelings"

Disagree strongly	1	
Disagree		2
Neutral	3	
Agree	4	
Agree strongly	5	

=====
PTB2

I think I could "beat" a lie detector.

/"Beat lie detector"

Disagree strongly	1	
Disagree		2
Neutral	3	
Agree	4	
Agree strongly	5	

=====
PTB3

I'm a rebellious person.

/"Rebellious person"

Disagree strongly	1	
Disagree		2
Neutral	3	
Agree	4	
Agree strongly	5	

2.2.3 Second Example of Specifications Based on the New Standard

This is the best example of a specification that was created in 2008 using the SRC Blaise Standards as a guideline. It reflects in a standardized fashion questionnaire content, screen design, programming, and analysis and documentation requirements. The programmer is given the Blaise field name, field tag, question text, interviewer instructions in the appropriate font and color, response options, skip instructions, data type information, and a field description.

HSC1_SC20

HSC1_SC20_NCS

The next questions are going to require you to think back over your entire life. Please take your time and think carefully before answering.

♦ **Read the following questions slowly**

Have you ever in your life had an attack of fear or panic when all of a sudden you felt very frightened, anxious, or uneasy?

/ "Had Fear or Panic Attack"

Yes..... 1 (GO TO HSC3_SC20_1)
No 5
DK
RF

Question Type: Enumerated; Range 1 and 5

HSC2_SC20a

HSC2_SC20a_NCS

Have you ever had an attack when all of a sudden

- you became very uncomfortable,
- you either became short of breath, dizzy, nauseous, or
- your heart pounded,
- or you thought that you might lose control, die, or go crazy?

/ "Had Other Attack"

Yes..... 1
No 5
DK
RF

Question Type: Enumerated; Range 1 and 5

HSC3_SC20_1

HSC3_SC20_1_NCS

Have you ever in your life had attacks of anger when all of a sudden you lost control and broke or smashed something worth more than a few dollars?

/ "Attack Broken Things"

Yes..... 1
No 5
DK
RF

Question Type: Enumerated; Range 1 and 5

HSC4_SC20_2

HSC4_SC20_2_NCS

Have you ever had attacks of anger when all of a sudden you lost control and hit or tried to hurt someone?

/ “Attack Hurt Someone”

Yes..... 1 (GO TO HSC6_SC21)
No 5
DK
RF

Question Type: Enumerated; Range 1 and 5

3. SRC Blaise Standards

The benefits of implementing the SRC Blaise Standards include a significant increase in instrument quality and a reduction in development time. By standardizing the specifications for Blaise survey instruments, we have also created an important and powerful communication tool for the survey manager, the specification writer, the client or researcher, the CAI programmer, and the testing team to use during the application development and testing phase.

Some of the basic requirements of the standardized specification include:

- Field Name, Field Tag, Field Description
- Question Text
- Interviewer Text
- Response Categories
- Routing/Skip Instructions
- Data Types

Other important elements of a Blaise specification are:

- Fill Logic
- Edit Masks
- Soft Consistency Checks (Signal)
- Hard Consistency Checks (Check)
- Explicit Checkpoints
- Logic for Constructed Variables or Re-Coding

3.1 General Specification Rules

Some general rules were developed for users to follow when specifying a question for a Blaise survey application:

Specifying A Blaise Question (SRC 2008)

For each question in a Blaise instrument, generally the following are specified;

1. Field Name (variable name), generally alpha-numeric (e.g. A1, B2, etc.). Note:
 - Avoid the use of underscores if possible, since field names are used in the Blaise program code and underscores increase programming time;
 - However, “ENTER all that apply” field names require underscores at the end (e.g., B1_);
 - Avoid the use of letters and numbers in positions where they may be confusing (such as the lowercase letter “l” and the number “1,” letter “O” and number zero “0,” the number “2” and the letter “Z,”; and

- Due to variable naming constraints with some data processing software, **Field Names should not exceed 16 characters**
- 2. Field Description, a brief meaningful descriptive text (e.g., Current 2. Grade), **not to exceed 25 characters in length**; this is what is displayed next to the entry window on the Blaise screen.
- 3. Question text
- 4. Response option categories (if an enumerated question)
- 5. Skip (go to) instructions
- 6. Data type (if not an enumerated question)

Note that a Blaise Field Tag will always be programmed; if not specified, it will be the same as the Field Name. As with Field Names, Field Tags have no spaces. Some surveys may need an additional variable descriptor or ID, and could use the Field Tag for this purpose.

MQDS can display the Field Name, Field Description, or Field Tag as the variable name or ID, thus providing flexibility in generating documentation for a variety of purposes.

Particular questions may have additional specifications:

- | | |
|---|---|
| 1. Online help indicator | 7. Explicit programmed checkpoint |
| 2. Interviewer instructions, including probes | 8. Routing instruction logic |
| 3. Optional text and variable text | 9. Logic for constructed or recoded variables |
| 4. Field-specific attributes (e.g., DK, RF, or EMPTY) | 10. Soft consistency check (Blaise SIGNAL) |
| 5. Edit mask | 11. Hard consistency check (Blaise CHECK) |
| 6. Logic for fills (variable text) | |

3.2 Sample Specification from SRC Blaise Standards

NOTE: Standard screen design icons and instruction bullets and colors are not required in the specifications

Field Name	{Alphanumeric, e.g., B1; no spaces}
Field Tag	{= Field Name if not specified; no spaces}
n of n	{e.g., 1 of 2; for multi-part questions, such as time-unit and period}

[F1] - Help

RB Page #
 Calendar
 Interviewer Checkpoint
 Question text ^fill (optional text)?
 Interviewer instruction(s)
 ENTER all that apply [if applicable]
 /"Field Description" {Meaningful description with spaces}

Name1	Response option1 label	1	GO TO NextQ
Name2	Response option2 label	2	GO TO NextQ
Name3	Response option3 label	3	GO TO NextQ
Name4	Response option4 label	4	GO TO NextQ
Name5	Response option5 label	5	GO TO NextQ
Other	Other – Specify	7	
DK			GO TO NextQ
RF			GO TO NextQ

[Enumerated, (implicit); requires response options and relevant skips, as above]

Integer; range n-n; edit mask

Numeric; n decimal places, range n.nn-nn.nn; edit mask

Currency; n decimal places, range n.nn-nn.nn; edit mask

String; width= n; Edit Mask

Open End

Attributes: [DK, RF, EMPTY, NODK, NORF]

Routing instruction logic

Constructed variable or recode logic

Fill logic

Condition, "Fill text"

Condition, "Fill text"

Soft consistency check:

Condition

"Probe text"

Signal number; e.g. "Signal Fieldname"

Hard consistency check:

Condition,

"Probe text"

Check Number; e.g. "Check Fieldname"

Programmer Notes

Adding a visible break between questions helps programmers to clearly see what specifications are associated with particular questions

3.3 Quick Reference Guide

As part of the SRC Blaise Standards, a "Quick Reference Guide" was created. Based on a concept demonstrated to us by Mathematica Policy Research, Inc., we developed a "Quick Reference Guide" that would provide the user with an at-a-glance view of a variety of common question types used in SRC Blaise survey instruments.

For each question type, we used a four-panel style to show the user the following:

- How the question is specified;
- How the question is programmed in Blaise;
- How the question appears in the Blaise Data Entry Program (DEP); and
- How the question appears in the Michigan Questionnaire Documentation System (MQDS) output.

3.3.1 Sample of Quick Reference Guide – Enumerated Question Specified

Yes/No with Skip Logic

B1
Do you currently use a computer, either at work, at home, or at school?

/ "Currently Use a Computer"

Yes	1	
No	5	GOTO B3
DK		GOTO B3
RF		GOTO B3

B2
Have you used the Internet?

/ "Used the Internet"

Yes	1	GOTO B4
No	5	GOTO B4
DK		GOTO B4
RF		GOTO B4

Specification

3.3.2 Sample of Quick Reference Guide – Enumerated Question Programmed

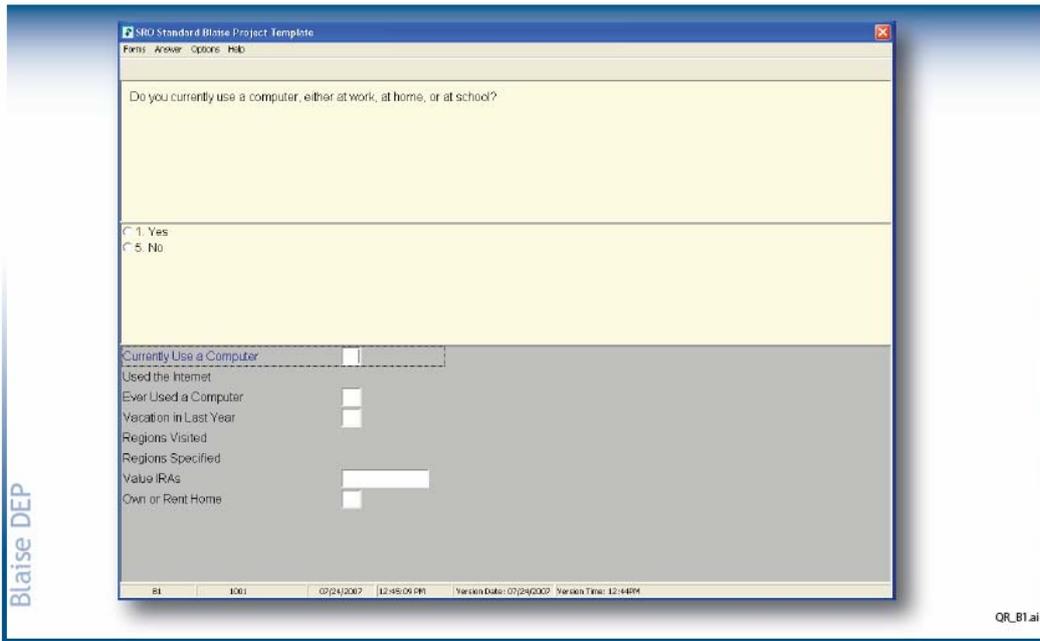
B1 (B1)
"Do you currently use a computer, either at work, at home, or at school?" /
"Currently Use a Computer" :

TYesNo

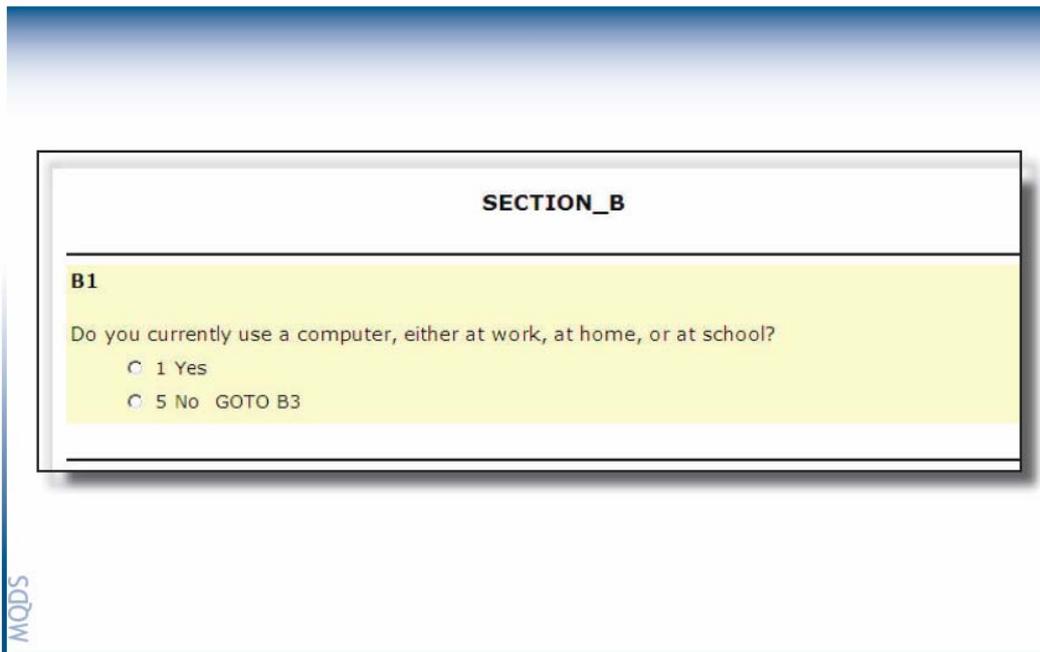
Programming

TYesNo :
(Yes (1),
No (5))

3.3.3 Sample of Quick Reference Guide – Enumerated Question in Blaise DEP



3.3.4 Sample of Quick Reference Guide – Enumerated Question in MQDS



4. Future Impact of SRC Blaise Standards

We believe there is sufficient anecdotal evidence that there is a significant positive impact on Blaise application development due to the introduction of the SRC Blaise Standards publication.

A future idea is to develop a proposal seeking funding to design and carry out an experiment to measure the impact of the SRC Blaise Standards on:

- Development time
- Development cost
- Application quality

We believe the SRC Blaise Standards in conjunction with the SRC CAI Testing Tool (CTT) (Dascola, Pattullo, and Smith 2007) has improved the overall quality and robustness of our Blaise survey instruments.

5. References

Couper, Mick P. 2000. *Development and evaluation of screen design standards for Blaise for Windows*. Paper presented at the 6th International Blaise Users Conference, Kinsale, Ireland, May.

Survey Research Center. 2008. *SRC Blaise Standards*. Ann Arbor: The University of Michigan.

Dascola, M., Pattullo, G., and Smith J. 2007. *CTT: CAI Testing Tool*. Paper presented at the 11th International Blaise Users Conference, Annapolis, Maryland, USA, September.

The Effects of Relocation on Blaise Support in O.N.S.

Lucy Fletcher, Office for National Statistics

1. Introduction

Until 2006, the Blaise support team (Blaise Development, Standards and Support, or BDSS) at Office for National Statistics (ONS) was based in London, within the Social Survey Division, which is responsible for carrying out social surveys such as the Labour Force Survey and the General Household Survey.

In 2006, ONS relocated many of its functions from London to Newport in Wales, including Social Survey Division (SSD) and as part of this, BDSS. Although everyone concerned had the opportunity to relocate, very few people moved and nobody from BDSS relocated. There was a big recruitment exercise to fill in the vacant posts across the whole office, and although many posts were replaced, much knowledge and expertise was lost. Many areas suffered and not just SSD - Labour Market Division, Business Surveys, Regional Analysis, as well as the central support areas such as Human Resources, Finance and IT and many others were all affected.

Prior to relocation, the solving of simple bugs, programming new questions and so on would be done by researchers within the survey. The loss of knowledge within a survey about their Blaise questionnaire, as well as the loss of general Blaise knowledge meant that BDSS were now doing more bug solving and programming. This extra burden on BDSS meant that they had little time to devote to development projects, such as exploring Blaise IS.

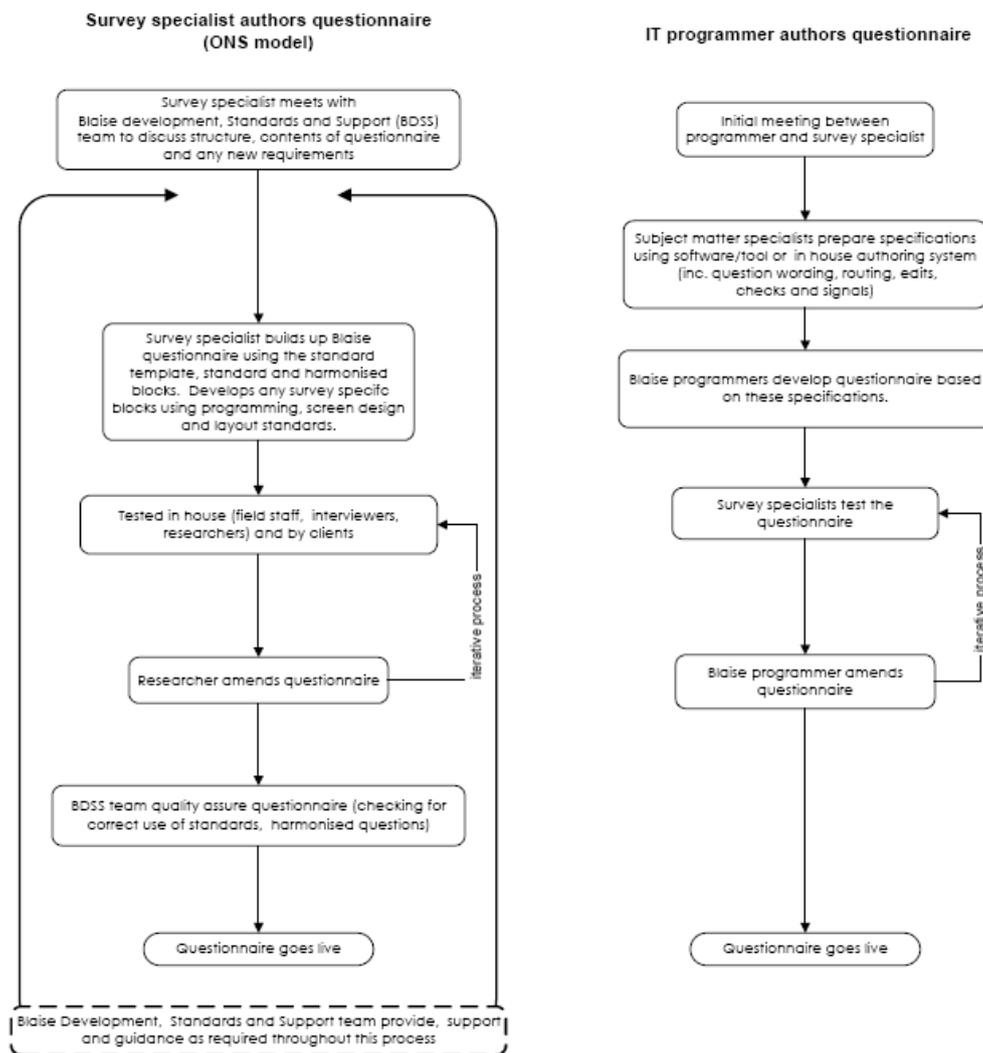
2. Prior to relocation

Many organisations have a team of specialist programmers who carry out all Blaise work, but in ONS, the programming is carried out by the survey researchers. The researcher is responsible for the authoring of the entire questionnaire (following what Gatward, 2007, calls the 'survey specialist model' as shown in Chart 1 below). ONS considers that it is important for survey quality that researchers have a hands-on control and knowledge of the questionnaire.

Prior to relocation, the BDSS team worked closely with the IT division, which is responsible for case management systems, telecommunications and data output. The team also had close links with the Field division which is in charge of interviewer recruitment, survey allocation and interviewer training.

Due to the stability of staff within SSD, BDSS team members and researchers within survey teams had built up their experience in Blaise as well as having an in depth knowledge of their survey questions and outputs.

Chart 1. Summary of approaches to authoring Blaise questionnaires



2.1 Training

In order to build up Blaise knowledge of new researchers when they joined SSD, BDSS ran a half day introduction to Blaise. The aim was to introduce people to the BDSS team and its role, as well as to introduce the basics of Blaise, including the chance to create a very small, simple questionnaire. Then shortly after, new researchers would attend a three day training course conducted by Statistics Netherlands, hosted in ONS, London which started with Blaise basics and covered more advanced topics such as nested blocks and hierarchical lookups.

In addition to these formal training courses, new researchers would also be given on the job training by their fellow researchers within the survey team. A standards seminar was presented by BDSS, which covered programming standards and screen standards for ONS's social surveys.

2.2 Standards/BDSS role

Whilst the majority of programming was done within survey teams, BDSS still had a role to play in assisting surveys in developing their Blaise questionnaires. BDSS provided a series of standard block templates to researchers to make programming easier and to keep surveys standardised, or 'harmonised'. ONS surveys have a series of demographic questions which should be the same across all government social surveys, such as household details, marital status, employment questions, ethnicity, tenure, benefits information and so on. This

ensures that data are comparable across surveys. Also there is a standardised administration block at the end of each questionnaire which includes calculation of outcome codes for response rates, as well as interview timings and notes. Any changes to these questions have to be agreed by the ONS Harmonisation team first, and then only BDSS can make changes. Survey teams then have their own blocks which they can amend as and when they need to.

BDSS also kept mode libraries and depmenus up-to-date for surveys to use. Quality assurance of surveys was carried out before each survey went out into the field, and a more in-depth audit was conducted each year to make sure that screen standards were being adhered to and that the surveys were using standard blocks and questions. New blocks and questions were tested thoroughly before being released to survey teams.

Researchers were able to handle most aspects of their questionnaires along with the use of block templates, and BDSS could just step in and help out where necessary, and assist with debugging and solving more difficult routing problems, and updating external lookup databases. BDSS operated an open-door policy where researchers could approach the team with queries or advice about their questionnaire, anything from creating a text file from an external lookup database to putting variables into a table.

BDSS had time to concentrate on development issues and looking at new ways of using Blaise-related tools (e.g. audit trails, Audio-CASI, event history calendars, documentation tools and so on). The team had time to develop new solutions such as rotation methods and also recode blocks and sections of code to make them more efficient.

The 'survey specialist model' was working well up to this point.

3. Addressing issues in the early phases of relocation

3.1 The relocation

Towards the end of 2004, ONS announced that it would be closing down the office in London and relocating most of its posts to its sister site in Newport, and the rest to its other site in Titchfield, near Southampton. Several hundred staff would either need to relocate or lose their jobs as part of a Government-wide cost-cutting exercise.

As many people decided not to relocate, the recruitment team in ONS had a huge job to build up the office in Newport. The relocating business areas of the office suffered from a huge loss of skilled staff and had to train up many new staff over a short period of time and continue to produce high quality outputs.

3.2 Effects on SSD

As with other business areas of ONS, the effect on SSD was the loss of a huge amount of knowledge and experience.

A system was put in place for around a year during relocation called 'parallel running' where a post was created and filled in Newport for each of the posts to be moved from London. The London staff who decided not to relocate trained up their 'shadow' member of staff to eventually take over their job.

3.3 Effects on BDSS

During relocation, BDSS was still in London but the people they were supporting were based in Newport. This led to issues usually associated with cross-site working such as poor communication and struggling without help. Eventually one person moved into BDSS in Newport from another work area as a 'shadow' member so the researchers had someone they could talk to directly and discuss their problems. There was very little Blaise experience in teams as most new recruits had not used Blaise before. The BDSS team in London finally left ONS early in 2008.

The effect of going through this major organisational change was that BDSS lost a lot of historical knowledge, such as why things were done in a certain way, and technical knowledge in that new members of the team had little or no Blaise knowledge. Questionnaires were not being quality assured before going out in the field, a task which used to be performed before each monthly or quarterly 'scatter' and then annually, in more depth. Standard blocks were being changed by survey teams who didn't realise the importance of making changes without notifying BDSS first, and so standardisation across surveys was lost. Testing of blocks and new questions was minimal, due to reduced resource in BDSS, and survey teams' lack of understanding of the importance of testing.

3.4 Breakdown of communication

BDSS used to have a good working relationship with the IT and Field divisions, which was facilitated by the fact that all three were based in the same office. The three areas worked closely to ensure that the entire survey process went smoothly, from questionnaire writing and testing to scattering and then to data production. Any potential issues that impacted on the process were discussed and resolved.

Relocation meant that Field staff were relocated to ONS's other site in Titchfield, with many staff choosing to leave rather than relocate, and communication diminished. Some IT staff were relocated to Newport, while some stayed in London. Working from three locations led to issues where new members of staff in each location were unsure of who did what and there was little face to face contact. There have been a few issues over responsibility, for example whether SVS or IT are responsible for the Manipula scripts for data production.

3.5 Other issues

Around the time of relocation, a major change was happening to one of ONS's surveys. The General Household Survey became a longitudinal survey and brought with it the extra complexities associated with this type of survey, such as feeding forward data and keeping track of respondents over time (see Setchfield, 2007).

Also at this time, ONS introduced the Integrated Household Survey (see Fiacco, 2007).

The ONS then added a couple of other new longitudinal surveys to its portfolio, namely the Household Assets Survey and later on, the Life Opportunities Survey (originally known as the Longitudinal Disability Survey of Great Britain).

With these complex additions, BDSS and survey team resources were really stretched.

4. What we've done and remaining challenges

4.1 New strategic direction

BDSS acknowledged that there was an issue with the provision of Blaise support in the division, as a result of relocation. The 'survey specialist' model that Gatward (2007) described was starting to break down due to the intense period of organisational change. She explained how it was necessary to constantly evaluate how the model is working, especially through change.

The team consulted with Blaise users and senior management and produced a strategy paper which presented different ways of working. At this stage every single option was explored, and this included taking away programming from survey teams altogether and giving all responsibility for this to the BDSS team.

The option chosen was for BDSS to take more responsibility over Blaise questionnaire development and programming for survey teams. It was proposed that BDSS have responsibility for programming all questionnaires for new surveys, with the survey development team providing specifications to BDSS. The survey team will be required to fill in the fields sections, design and implement (with assistance) signals and checks, and program rules if a block is straight forward. In essence, BDSS will provide a skeleton, with any functionality (such as rotation, person selection) required.

Once a survey is in the field, survey teams will be responsible for maintaining the field section, checks and simple routing. BDSS will still undertake complex routing or implementing complex logic into the Blaise questionnaire. Survey teams would still be responsible for coordinating testing and any queries relating to the Blaise questionnaire. However, BDSS will develop best practice principles in relation to questionnaire testing and provide any tools or templates. Survey teams would still be responsible for interactions with IT including scattering of the questionnaire.

The overall aim is to reduce the amount of expert programming required by survey teams, whilst still giving them control and flexibility with their instruments. By centralising the complex programming, it should mean less training overall, and less reliance on a particular survey team member.

4.2 Staffing levels

Following the strategy paper, the BDSS team has expanded from one person to five full time members of staff (in London it was usually around four to five members of staff with a mixture of full and part time). Two members of the team are survey researchers. Following a period of on the job coaching and training, all members of the team are now skilled in Blaise and Manipula and have a combined knowledge of all of ONS's social surveys.

Also the team has recently secured continuous help from the Business Operations Support area which works alongside the survey teams, providing administrative support and dealing with customers. They help BDSS out with organising the Blaise training and creating interviewer instructions using Robohelp software.

The extra resource will allow the team to take on extra programming from survey teams where necessary, and allow the team to concentrate on other areas such as development projects and implementing new ideas and tools.

4.3 Training

The training that BDSS provides to new members of staff has not changed at all since before relocation. New starters still attend a half day course which the team runs internally, then a three day course run externally. Coaching within teams by experienced researchers is not happening as much as it should, which is what has always kept the 'survey specialist model' working. Perhaps with hindsight we should have looked at changing our training to meet the needs of so many new staff, although it was difficult to do this with limited resource. We will be actively encouraging on the job coaching within survey teams, so that knowledge isn't confined to one particular team member, which is an issue faced by many teams at the moment.

We realised that communication had broken down between BDSS, and our Field staff and IT division as an effect of relocation. We have now set up regular meetings to discuss issues and problems, and also a series of training sessions and question and answer sessions so we can understand how each other operates, who has responsibility for what areas such as writing Manipula scripts, and to create good working links.

We already have a special questionnaire set up called an 'Electronic Learning Questionnaire' which is used to train new interviewers and help them to become familiar with the survey they are working on, and how to handle certain interviewing situations. As part of our training overhaul, we will be introducing an ELQ for new researchers to help them understand the basics of how Blaise works, and to demonstrate screen standards.

4.4 Support strategy

During relocation, it was difficult to keep up with records of changes to questionnaires, and other documentation. The interviewer help system was very out of date and there wasn't enough resource to keep it updated with the latest changes, which the interviewers found very frustrating. All Blaise users found it difficult to find out what changes had been made to questionnaires and the reason for any changes.

The BDSS team is about to introduce a new system of logging support work, and this will involve survey teams logging their problems into a database wherever possible before seeking help from our team. This is to encourage researchers to think through their problems first as we believe many issues can be resolved when they start to write out and think through the problem. The team will look over the types of issues that are occurring and make sure that they are covered by our training program. This method will free up time for BDSS to concentrate on other non-support work.

4.5 Reintroduction of QA/Standards

One of the effects of relocation was the loss of the quality assurance of surveys before they go into the field. We will be reintroducing this, and will make sure that surveys are all using the correct versions of the standard blocks and the specified screen layouts. Desk instructions and training will be improved to make sure that survey teams are aware of what versions of the blocks they should be using and how to apply the correct layouts.

BDSS will have time to make sure that blocks are thoroughly tested before being released to survey teams which will reduce the chances of errors creeping into questionnaires. For example, some routing was accidentally changed recently in a standard block which meant that one of the harmonised tenure questions was only being asked to the respondents of one survey instead of all surveys. This affected around two months of data until the error was spotted and fixed. Normally most errors can be fixed using Manipula, but it was unfortunate that this prevented the data from being collected at all. Part of the problem was that some users had modify access to the standard blocks on the network and didn't realise that a change would affect other surveys (questionnaires directly reference these blocks on the network). Access control has been tightened now to prevent mistakes like this from happening again.

4.6 Documentation and specifications

Before BDSS takes on any programming task, we now make sure that the research teams provide us with a detailed specification of the questionnaire first, and any flow charts where necessary. This allows the researchers to think through their problems and the issues and makes our jobs as programmers much easier. Too many programming tasks that we have been given have been poorly described, and BDSS members often end up giving advice on areas such as question design.

One example of a recent problem with a survey was when BDSS was asked to help out with some routing. The survey team was asked for a paper questionnaire, and was provided with automated documentation taken directly from the questionnaire. This included the errors that needed to be fixed. Many surveys had been working from this type of documentation rather than keeping their original client-specified paper questionnaire up to date. This made it hard to fix errors as it was impossible to see what the questionnaire routing should look like. Having time to look at documentation tools should help to stop this from happening again.

5. Conclusion

Following the relocation of SSD and BDSS to Newport, it was acknowledged that a great deal of survey knowledge and Blaise expertise was lost, and the traditional approach to Blaise support was not working very well.

After consultation with users and senior managers in SSD, the BDSS team produced a report which outlined all issues surrounding Blaise support in the division, and then successfully managed to recruit more staff into the team. New ways of working were implemented such as a new support strategy, and reintroduction of quality assuring and better maintenance of standard blocks. The approach to training was revised, and a new approach to improve communication between IT and Field was devised.

SSD decided that the researcher doing the programming is still the best model to use in ONS, rather than having BDSS doing all the programming. This model has however been modified post relocation so that BDSS can take on more programming tasks. BDSS will monitor how well the new changes are meeting the needs of the survey teams.

6. References

Fiacco, A. The ONS Integrated Household Survey: The next instalment. Proceedings of the 11th International Blaise Users Conference, 2007, Annapolis.

Gatward, R. Authoring Blaise questionnaires - a task for the survey specialist or IT programmer? Proceedings of the 11th International Blaise Users Conference, 2007, Annapolis.

Setchfield, C. Coping with people who just won't stay put: The use of Blaise in longitudinal panel surveys. Proceedings of the 11th International Blaise Users Conference, 2007, Annapolis.

Evaluating the Use of Questions and Responses in a Large National Dietary Data Collection Instrument

Lois Steinfeldt, John Clemens, and Jaswinder Anand, United States Department of Agriculture, Agricultural Research Service, Beltsville Human Nutrition Research Center, Food Surveys Research Group

1. Introduction

The U. S. Department of Agriculture's Automated Multiple Pass Method (AMPM) Blaise instrument collects 24-hour dietary recall data for the What We Eat In America (WWEIA), the dietary interview component of the National Health and Nutrition Examination Survey (NHANES). AMPM contains over 2,400 questions and more than 21,000 responses about foods with each response determining the next appropriate question. This results in approximately 500,000 possible paths through the questions within the AMPM. Each year it is used in approximately 9,000 interviews which ask individuals to recall the foods and beverages that were consumed the day before the interview. On average, 13 foods are reported for each 24-hour dietary recall and 10 questions are asked about each food. Of the 10 questions, 6 are probes about the description of the food and the amount consumed. During each year of the survey, interviewers use AMPM to ask respondents over a million questions about foods.

AMPM has been in continuous use since January of 2002 and in that time has been used to conduct over 60,000 dietary intake interviews. A large comprehensive update that includes major procedural changes is done for the beginning of each two-year survey period. A smaller update incorporating primarily new foods and portions is done as necessary for the second survey year. Updates are made to questions, response values, and to programming which controls what questions are asked based on previous responses. The purpose of the updates is to assure validity and completeness of the dietary data. These updates result from changes in the food supply and in food consumption patterns, and evaluation of the use of the questions and responses.

2. Qualitative methods

Qualitative evaluation methods are an important part of all phases of data collection. At the Food Surveys Research Group (FSRG), the quality assurance procedures are based on years of experience in identifying and eliminating the source of errors and inconsistencies that occur in food and nutrient intake data. Fortunately some past sources of errors, such as missing responses, have been eliminated with the automation of the collection method. Comprehensive training and monitoring programs for interviewers complement the automated system to help ensure that data collected are of good quality. Data collection oversight is provided for the dietary interviewers by FSRG staff. This includes direct observation of interviews, interviewer questionnaires, review of data collected, and informal feedback to interviewers. New interviewers complete a rigorous training program before starting work. Because of the changing food market and emerging nutrition issues, changes are made to the AMPM each year. These changes, as well as the need to assure standardization of data collection procedures, necessitate periodic refresher training sessions for the dietary interviewers. Training is conducted just prior to the launch of each year's data collection. On-site visits are conducted to observe the data collection and to provide technical oversight. The interviewers are monitored four times each year. During these field observations, interviewers are evaluated on the following:

- Following interviewing protocol for the survey
- Proper use of automated system
- Overall demeanor, rapport, and pace

Randomly assigned interviews are monitored by supervisors for quality assurance purposes.

3. Quantitative methods

Statistical analysis of the use of questions and responses provides a quantitative approach to evaluation which is particularly useful with a large and complex instrument such as AMPM. SAS 9.2 was used to statistically analyze the data. Results from four years, 2005 to 2008, were combined to produce over 2.8 million responses to questions in the AMPM about food details and amounts from more than 35,000 intakes. In 2008, AMPM contained 2,305 questions about food details and food amounts. In addition to these questions, there were 130 questions used to record “Same As” foods. AMPM allows the interviewer to capture and record a food as the same as one eaten previously in the day or the same as a food eaten by another household member. In these cases, AMPM does not prompt for the food detail questions again, but does prompt for the amount of the food eaten. This reduces the number of repetitive questions during an interview for frequently consumed foods such as coffee and when conducting interviews for members of a household who all ate the same dinner.

Table 1 shows the results for how many of the total questions in AMPM in 2008 were used during the 2005 to 2008 time period. Overall, 90% of the questions were used, with a higher percentage of food detail questions used than food amount questions. Overall, this is a very high percentage use of questions in an instrument which obtains detailed information on foods and amounts of food eaten in a large and diverse population with a wide-ranging and constantly changing food supply.

Table 1. Use of AMPM food detail and amount questions

	Count of questions	Questions used in 2005-2008 Number (Percent)
Food detail	950	909 (96%)
Amount	1,355	1,215 (90%)
Total questions	2,305	2,124 (92%)

Most of the questions in AMPM have the option to enter responses not on the list. These are designated as “Other, Specify” (OS) responses. The questions without this option are usually numeric or yes/no type questions. There were a total of almost 44,000 OS responses in the 4-year time period which is an average of about 1.2 OS responses per intake. OS responses are particularly important in a continuing national food consumption survey. They record and document new foods and new package sizes in the marketplace which may then be added to the AMPM. They are also used to collect unusual foods which are consumed infrequently by the U. S. population.

Of particular interest in evaluating the use of AMPM is the percentages of “Don’t know” responses. Almost all the question in AMPM allow a “Don’t know” (DK) response. There were a total of almost 60,000 DK responses in the 4-year time period which is an average of 1.7 per intake. The questions in AMPM with higher DK counts are food characteristics which are of interest, but may only be known to the person preparing the food. An example of this is the question “What percent lean was the ground beef?” One of the factors that make this question difficult for respondents is changes in labeling over the past decade. AMPM interviewers are trained to accept “Don’t know” responses rather than encourage the respondent to guess. Food and nutrient composites are used as appropriate for analysis for foods with DK responses.

Table 2 shows the percent of questions with at least one DK or OS response for each year. Overall the percentages show very little change year to year. There does seem to be a slight downward trend in both DK and OS responses over the 4 years.

Table 2. Percent of questions with at least one DK or one OS response by year

	2005	2006	2007	2008
Don't know (DK)	43.5%	43.5%	43.4%	43.4%
Other, specified (OS)	43.6%	42.4%	41.4%	40.8%

Table 3 shows the percent of all responses that are DK or OS for each year. These percentages show a slight downward trend. The Cochran-Armitage test was used to test for trends in DK and OS responses over the 4 years. The null hypothesis for the Cochran-Armitage test is that there is no trend. One-sided and two-sided p-values are computed for the trend test. The right-sided p-value tests for increasing trend in the proportions and the left-sided p-value tests for decreasing trend. Overall, both DK and OS responses showed statistically significant decreasing trends with p-values <0.001.

Table 3. DK and OS as a percent of all responses by year*

	2005	2006	2007	2008
Don't know (DK)	1.9%	1.8%	1.6%	1.6%
Other, specified (OS)	1.4%	1.3%	1.2%	1.1%

* Cochran-Armitage Test for Decreasing Trend $p < 0.001$ for both DK and OS.

While the overall numbers show very little change, there could be larger changes in individual questions. With the large number of questions in AMPM, checking the percentages of DK and OS responses for each question for each year would be very time consuming. Statistical tests of differences in the response percentages for DK and OS for the same question over the 4 years would provide indicators of where changes are occurring that need to be evaluated.

Of the 2,305 questions, 1827 were selected that were used in all 4 years. Of these, 713 (39%) had no DK responses over all 4 years and were eliminated from the DK analysis. 828 questions had no OS responses over all 4 years and were eliminated from the OS analysis. Table 4 shows that only two percent of the questions showed significant increases ($p < .05$) in the numbers of responses for both DK and OS. Eight percent of questions showed significant decreases ($p < .05$) in the number of DK responses and twelve percent of questions showed significant decreases ($p < .05$) in the number of OS responses. Most of the questions showed no significant trend for both OS (86%) and DK (90%) responses testing at $p < .05$.

Table 4. Results of trend tests for OS and DK responses

	Questions showing no significant trend	Questions showing significant decrease ¹	Questions showing significant increase ²	Total number of questions
Don't know (DK)	1,005 (90%)	87 (8%)	22 (2%)	1,114
Other, specified (OS)	864 (86%)	116 (12%)	19 (2%)	999

¹ Cochran-Armitage Test for Decreasing Trend $p < 0.05$.

² Cochran-Armitage Test for Increasing Trend $p < 0.05$.

Using the statistical tests reduced the number of questions to be reviewed for DK and OS responses from 2,305 to 22 for DK and 19 for OS. Questions which showed decreasing or no trend, do not warrant evaluation related to these responses. There were no questions with statistically significant increases in both the numbers of OS and DK responses. Of the 22 questions showing increases in DK responses, 16 are food detail questions and 6 are amount questions. Of the 19 questions showing increases in OS responses, 14 are food detail and 5 are amount questions. Although these questions show a statistically significant upward trend in DK or OS responses, the total number of DK or OS responses remains small.

Statistical tests can also be used to look for changes in the percentages of existing response options. For example, the responses to the question "What kind of milk?" over the 4-year time period can be tested for statistically significant changes using Chi Square. Table 5 shows the percent of responses by year. The Chi Square test for differences in the distribution of responses over time is significant at $p < .001$. The percentages in table 5 show a shift to lower fat milk. Of particular interest is that there is very little differences in the percentages of soy, other and don't know responses. In this case the statistical test confirms a known trend in U. S. food consumption patterns. This same approach can be used for other questions, to look for other food consumption and /or food supply patterns in the data.

Table 5. Percent of responses to “What kind of milk?”*

Milk Kind	2005	2006	2007	2008
Whole milk	39	37	32	34
2% milk	37	36	39	39
1% milk	9	11	11	11
Skim/nonfat milk	9	10	10	10
Soy milk	2	2	2	2
Other	2	2	2	2
Don't know	2	2	2	2
Total	100	100	100	100

*Chi Square significant $p < .001$.

4. Conclusion

Continual review and update of the AMPM is required to reflect important changes in the U.S. food market such as new foods, ethnic foods, and new package sizes. Changes also are directed by the data collected in WWEIA and to address current public health concerns. With over 2,400 questions and more than 21,000 responses, it is not possible, nor is it needed, to review all the questions and possible responses on a yearly or bi-yearly basis. Statistical tests can be used to identify questions where the percentages of all the response options including “don’t know” and “other, specified” may be changing over time. While changes in questions may be statistically significant, infrequently used questions and questions with low percentages of DK and OS responses may not be significant in terms of the food and nutrient data produced by the survey. However, as shown in this analysis, this approach can be used with other methods of prioritizing instrument review to substantially reduce the number of questions to be evaluated and focus scarce resources productively.

Different Methods of Working with Blaise® in the

Israel Central Bureau of Statistics

Shifra Har and Evgenia Luskin, Israel Central Bureau of Statistics

1. Introduction

The Israel Central Bureau of Statistics (CBS) is using the Blaise software for more than 10 Years. This software is used in the development of many surveys – especially in the arena of families' surveys and individuals' surveys. More than 20 systems have been developed using the Blaise software. In particular I'd like to mention that Blaise was used to develop the CBS population and housing census questionnaire that was conducted in 2008.

Some additional surveys that use Blaise include the labor force survey, the family income survey, the social survey, the health survey, the first degree graduates survey, the crime survey and more.

At first we have used the Blaise tool for questionnaire development and Manipula for the survey management systems. We overcame the problems we had with the Hebrew language, which is written from right to left, in developing the surveys. During time we have also found solutions to the problems of keying-in two additional languages: Arabic and Russian. Today the questionnaires for all of our surveys are allowing enumeration in these three different languages.

As time went by the developers work procedures have changed, new tools, software development languages and databases were introduced – so we combined the use of several tools to get the maximum benefit from each tool. Despite all of the above, we still keep the Blaise as the main tool for developing surveys.

Massive work with the Blaise software gave us the ability to gain huge experience in using the different Blaise tool components. With time, the computerized Blaise questionnaires became more complex and combined different modes of data collection: CAPI (Computer-assisted personal interviewing), CATI (Computer-assisted telephone interviewing) and CADI (Computer-assisted data input). These kinds of complex surveys demanded more complex management systems as well.

We are about to start the development of a system for generic management of surveys that include different modes of data collection. The CBS decided to use Blaise for developing the questionnaires for all of the enumeration techniques for this generic survey management system.

The CBS puts a main emphasis on information security. One of the issues we are handling is the writing of secure code. The development methodology of the CBS demands that we put special attention to the fact that the enumerators are sending us information from all over the state of Israel through the internet network which is an unsecured network. This means that we must use encrypted data to transfer the survey information. It is not enough to develop safe communication; additional steps are taken in each individual computer so that the software code and the data itself are secure. "Information noise" is entered into the data, so if anyone is listening they will not be able to decipher anything.

We can distinguish between several methods of working with the Blaise tool when developing computerized surveys:

1.1. The "**Blaise + Manipula**" method

The first method is using the Blaise and Manipula in a natural way. The questionnaire is written using Blaise, the management system is written using Manipula and the data is stored in the Blaise database.

1.2. The "**Blaise + .Net**" method

In this method the questionnaire is written using Blaise, the management system is written using .Net, the questionnaire data is stored in the Blaise database and the survey management data is stored in an SQL server database.

1.3. The ".Net + SQL" method

In the last method the questionnaire is written using Blaise, the management system is written using .Net, both the questionnaire data and the survey management data are stored in the an SQL server database.

Each of the above three methods has its own advantages and disadvantages. Before each new survey there is an examination of which method is the most suitable for that survey.

2. The different methods of working with Blaise

In the following sections I will go into detail and give examples for surveys that were done in each one of the three methods. I will note the advantages and disadvantages of each method as well.

In general, the CBS has been conducting surveys in all three methods, but in recent years the direction is to develop more systems with .Net (the last two methods) and less with Blaise only (the first method).

2.1. The "*Blaise + Manipula*" method

Historically, this is the first method that was used. The CBS labor force survey, social survey and health survey were written 100% in the Blaise tool. The survey management system in the survey gathering center and in the laptops was written in Manipula.

Advantages of the "*Blaise + Manipula*" method:

The Blaise language enables quick development which helped us move quickly from paper surveying to computerized surveying. The systems have proven their stability and efficiency.

Disadvantages of the "*Blaise + Manipula*" method:

Manipula is limited as a development tool. The limitations are especially noticed in the screen designs, batch activities in the management system and multi users' capabilities.

We still use the "*Blaise + Manipula*" method for relatively small and simple surveys. This means surveys that do not use the CAPI mode of data collection and there is one coordinator that controls the data gathering.

Examples for current usage of this method are the first degree graduates survey and the public transport satisfaction survey. The first labor force survey that was developed in this method has already been replaced with a more advanced system.

In the social survey we still use the old management system but the new management system is under development and will be in production in 2010.

2.2. The "*Blaise + .Net*" method

In this method the questionnaire is written using Blaise, the management system is written using .Net, the questionnaire data is stored in the Blaise database and the survey management data is stored in an SQL server database.

Starting from Blaise version 4.6 and with the help of the module BCP 2 we are combining Blaise and .Net in two ways; from Data Entry Program (DEP) to .Net and from .Net to DEP.

Examples for usage of this method:

2.2.1. From DEP to DLL that is written in .Net

In the questionnaire of the social survey the finding and marking of streets is done with DLL. In the questionnaire of the health survey the DLL checks the validity of key fields and inserts data into the questionnaire from an SQL management table.

In figure #1 in the appendix we can see two procedures:

- ❖ The procedure GetMezahe returns the output from the validity checks in the parameter p_TozaatPkida
- ❖ The procedure GetSqlData brings data from the SQL database to the export parameter list.

2.2.2. From .Net to DEP

In a management system that is written in .Net there are calls to DEP for surveying or questionnaire editing, for calls to Manipula programs and for different activities done with Blaise tables.

Advantages of the "Blaise + .Net" method:

Utilizing the strong abilities of the Blaise language which include quick development, ease of changing a questionnaire and stability. Using this with .Net gives us a solution for the screen design problems and gives us a wide variety of development options. Performance of activities with SQL tables is much faster than with Blaise tables so the management system is much more friendly and quick.

Disadvantages of the "Blaise + .Net" method:

The performance of activities using Blaise tables is relatively slow compared to SQL tables.

Examples for current usage of this method are the labor force survey which is already in use and the social survey which is still under development.

2.3. The ".Net + SQL" method

In this method the questionnaire is written using Blaise, the management system is written using .Net, both the questionnaire data and the survey management data are stored in the an SQL server database. During the survey only one questionnaire exists in the Blaise questionnaire table. After exiting from the questionnaire all of the data is transferred to an SQL table and deleted from the Blaise table.

Actually we are using Blaise as the GUI for the surveying screens. In this method we need to develop two modules. The first is for storing the Blaise data into SQL and the second is for uploading the Blaise questionnaire from data stored in the SQL. Both modules are written in .Net.

In figure #2 in the appendix we can see part of a management program that creates a Blaise table, fills it with data from an SQL table, entering DEP, exiting the questionnaire and saving the data in SQL and at the end deleting the Blaise table.

Advantages of the ".Net + SQL" method:

Using this method gives us great performance compared to the first two methods. It utilizes the Blaise strong abilities as a language for questionnaire development, the ease of changing a questionnaire and the stability of the DEP software.

Disadvantages of the ".Net + SQL" method

This method requires the creation of an SQL table for keeping the questionnaire data. Each change in the questionnaire like adding a question or changing the definition, requires a change in the SQL table as well.

The two additional modules that are required for transferring the data between the Blaise tables and the SQL tables are very large. There must be reference in these modules for each field and the structure of the fields should be identical. This requires a long development and debugging time.

The Blaise advantage of fast development is diminished with the need to update these additional modules with every change.

So the main disadvantages of this method are lack of flexibility and long development time.

A good example for the ".Net + SQL" method can be found in the 2008 CBS population and housing census.

The methodology for the population and housing census survey is a new methodology called the integrated census. This census is based on an integrated use of data from administrative files, together with sample data gathered in surveys, i.e., in the census field work of 20% of the population. For the questioning of 20% of the population, which is about 320,000 households, we used laptops that contained the Blaise questionnaire.

The questionnaire held approximately 300 questions and was comprised of three parts: questions about the apartment, questions about the household and questions about the individual. The questionnaire format is similar to the labor force survey questionnaire.

Approximately 2500 enumerators and coordinators participated in this survey. Two management systems have been developed, one for the enumerator and one for the coordinator. Due to the amount of users and the survey requirements the main criteria for the system was good performance – that is why the development method that was chosen is the ".Net + SQL" method.

Modules for data transfer between Blaise and SQL were written in C-Sharp. This enabled good performance in the laptop system and also in the management systems that were located in the regional offices and the national management system.

For the survey data two SQL tables were created; one for the survey answers and the second for different statuses: "response", "don't know" and "refuse". In figure #3 in the appendix we can see part of the software that was developed for the population and housing census in which the questionnaire is being filled from the SQL data and entered into DEP.

The objective of the developers of the census questionnaire was to create a surveying procedure for an enumerator that is not skilled. The enumerators training was very intensive but still many of the enumerators and coordinators didn't have experience working with laptops or didn't have surveying field experience before being recruited for the job.

This is why the questionnaire developers made the text of the questions very detailed with many comments for the enumerators. The enumerator had no freedom to make decisions based on experience; each question was very structured and written properly. The surveying was done in three languages; Hebrew, Arabic and Russian. Most of the text was not constant but was matched to second or third person according to the specific grammar of the chosen language. A strong emphasis was given to the logical checks in order to avoid wrong data entered into the questionnaire.

In this questionnaire of the population and housing census we made the first try of calling DLL from DEP for finding and marking of streets. From the perspective of development languages the flow was .Net -> Blaise -> .Net and this flow was not 100% stable. We got a few complaints from the field about a lost of focus in the street selection screen. There weren't too many such complaints, but we were not able to ignore this issue. The problem is that Blaise does not support work in a multi-threading environment. This is one of the lessons that we learned from the 2008 population and housing census.

The main disadvantages of the ".Net + SQL" method which are the lack of flexibility and long development time were not significant in the 2008 population and housing census since we had enough time allocated for development after the final changes were done to the questionnaire.

We found two other disadvantages to this method:

- ❖ Comments on fields are not downloaded from Blaise to SQL – so they are not used
- ❖ The received signal status is not stored in the SQL tables. So after uploading the data from SQL to the Blaise tables for further handling, the user is expected to re-approve signals that were already approved

3. Generic Management of Surveys

The number of different surveys that are conducted in the Israel CBS in the last few years has greatly increased. They were done in different modes of data collection: mail, field, telephone or a combination of the above. Internet surveys were introduced as well. The management systems for these surveys were written in Blaise + Manipula or in several other development languages. The main difference between these systems was the personal viewpoint of the person/department ordering the survey and the available resources for the project.

The purpose of the generic system for managing surveys is to reduce the resources needed to establish a new survey.

A strategic decision that was made in the generic management system is to separate between the questionnaire and the management system. Each questionnaire should contain a standard section with management data.

A generic survey will include the following sections:

- ❖ Survey definition and creation of required components
- ❖ Management for each of the different modes of data collection and integration between them
- ❖ Management of internal and external users
- ❖ Interface to the questionnaire
- ❖ Control and management tables
- ❖ Information security
- ❖ Archive management

The generic survey management system will use the "*Blaise + .Net*" method.

4. Summary

The three methods of developing surveys with Blaise each have their own advantages and disadvantages. The best development method for a specific survey is chosen according to the requirements of that specific survey, the different modes of data collection and other special requests.

The Israel CBS sees the Blaise as our main tool for questionnaire development. The development of a generic survey management system will allow managing a survey with efficiency and user-friendliness. In addition the interface between the management system and the questionnaire will be standard. This will reduce time and resources needed to develop new surveys.

Appendix

Figure #1:

```
PROCEDURE GetMezahe
```

```
parameters
```

```
import
```

```
p_Mezahe :string
```

```
p_Shana  :integer
```

```
p_Reva   :integer
```

```
p_Shavua :integer
```

```
p_Shlav  : T_Shlav
```

```
export
```

```
p_TotzaatPkida :Integer
```

```
Alien ('checkMezahe.GetMezahe','TotzaaBdika')
```

```
ENDPROCEDURE
```

```
-----  
PROCEDURE GetSqlData
```

```
parameters
```

```
import
```

```
p_Mezahe :string
```

```
p_Shana  :integer
```

```
p_Reva   :integer
```

```
p_Shavua :integer
```

```
p_Shlav  :integer
```

```
export
```

```
p_TaarSiumPkida : string
```

```
export
```

```
p_KnisaLeSviva : integer
```

```
export
```

```
p_StatusSheelon : integer
```

```
export
```

```
p_Ktovet      : string
```

```
export
```

```
p_ShemYishuv  : string
```

```
export
```

```
p_Mishpacha   : string
```

```
export
```

```
p_SugMidgam   : integer
```

```
export
```

```
p_PkidaHachnasot : integer
```

```
export
```

```
p_Merchav     : integer
```

```
export
```

```
p_MispMishkeyBait : integer
```

```
Alien ('checkMezahe.GetMezahe','FillFieldsFromSql')
```

```
ENDPROCEDURE
```

Figure #2:

```
'Creating instance of central BLAISE DB for rakaz
Dim InpDb, OutDb As String
InpDb = ConfigurationSettings.AppSettings("MaslulBikoret")
OutDb = "C:\\"
OutDb = OutDb & System.Environment.UserName & "\" & "ShBikoret"

objOutBikoret = objdatabaseManager.OpenDatabase("")
objOutBikoret.DictionaryFileName = InpDb & ".bmi"
objOutBikoret.DataFileName = OutDb & ".bdb"

Opening Blaise Db
objOutBikoret.Connected = True

*****

'Filling Fields of Blaise Questionnaire from SQL Server DB
MiluiSadotBeShelon(i, Par)
objOutBikoret.WriteRecord()

Closing Blaise Db
objOutBikoret.Connected = False

*****

Opening the Blaise Questionnaire for user
fl.Copy(InpDb & ".bmi", outDb & ".bmi", True)
fl.Copy(InpDb & ".bdm", outDb & ".bdm", True)
mystr = MaslulDepExe & " " & outDb & "/G /X /K" & par
id = Shell(mystr)
Dim myproc As Process
Dim ip As Boolean

'Checking whether the user got out of Blaise Questionnaire
myproc = System.Diagnostics.Process.GetProcessById(id)
ip = myproc.HasExited()
Do While ip = False 'Blaise is open
    ip = myproc.HasExited()
Loop
Return id

*****

'Saving changes from Blaise Questionnaire in SQL Server DB
objdatabaseManager = New BIAP14A2.DatabaseManager
objOutBikoret = objdatabaseManager.OpenDatabase("")
objOutBikoret.DictionaryFileName = InpDb & ".bmi"
objOutBikoret.DataFileName = OutDb & ".bdb"
objOutBikoret.Connected = True

If KeyValueExists(objOutBikoret, Par) Then
    objOutBikoret.ReadRecord()
End If
GetNetunimFromShelon(i)
SaveShelonInSql(i)
objOutBikoret.Connected = False

*****

Deleting Blaise DB
If Not fl.Exists(OutDb & ".-lk") Then
    fl.Delete(OutDb & ".bf")
    fl.Delete(OutDb & ".bjk")
```

```

        fl.Delete(OutDb & ".bpk")
        fl.Delete(OutDb & ".bsk")
        fl.Delete(OutDb & ".brd")
        fl.Delete(OutDb & ".bri")
        fl.Delete(OutDb & ".bdb")
        fl.Delete(OutDb & ".bmi")
        fl.Delete(OutDb & ".bdm")
    End If
    *****
    Refreshing data set with updated data from SQL Server DB and showing it to the User
    ds = exc.GetNetuneimFromBikoret(MyUser.Substring(5, 2), FlagSiyum, Shana_, Hodesh_)
    dtReshima = ds.Tables(0)
    grdReshLeBikoret.DataSource = dtReshima
    grdReshLeBikoret.Refresh()

```

Figure #3:

```

private void SetStatus2BIField(FieldStatus dsStatus, ref Field blField)
{
    try
    {
        string sField4Status = blField.Parent.IndexedName + "." + blField.LocalName;
        switch (dsStatus)
        {
            case FieldStatus.DoNotKnow:
                blField.Database.get_Field(sField4Status).Status =
                    BIFieldStatus.blfsDontKnow;
                break;
            case FieldStatus.Refusal:
                blField.Database.get_Field(sField4Status).Status =
                    BIFieldStatus.blfsRefusal;
                break;
        }
    }
    catch (WriteToLogException ex)
    {
        cWriteToLog.WriteToLogExceptionOnly(ex);
    }
    catch (Exception ex)
    {
        cWriteToLog.WriteToLog(ex, "");
    }
}

int indexPerson = GetIndex(fld.Name);
// new 2008
if((fld.Name.IndexOf("Butal.Men")>-1 || fld.Name.IndexOf("Hativa01S")>-1 || fld.Name.IndexOf("Hativa01D")>-1) &&
indexPerson<=drAll.Length)
{
    pdRow=(Packet.PersonsDataRow)drAll[indexPerson-1];
}
if(fld.Name.IndexOf("Hativa01A2")===-1 && fld.Name.IndexOf("Hativa01A")>-1 &&
indexPerson<=drResidentsArray.Length)
{
    pdRow=(Packet.PersonsDataRow)drResidentsArray[indexPerson-1];
}
if(fld.Name.IndexOf("Hativa01A2")>-1 && indexPerson<=drNonResidentsArray.Length)
{

```

```

        pdRow=(Packet.PersonsDataRow)drNonResidentsArray[indexPerson-1];
    }
    if(pdRow!=null && pdRow[strColumnNameInSql] != DBNull.Value)
    {
        if(fld.FieldDef.IsSet)
        {
            fld.TextAsSet = pdRow[strColumnNameInSql].
                ToString().Trim().Split(new Char[]{'-'});
        }
        else
        {
            fld.Text = pdRow[strColumnNameInSql].ToString().Trim();
        }
    }
    //set Status to Field(Blaise)
    if(pdRow!=null && dsHousehold.Tables[strTableNames[0]].Columns.Contains(strStatusFieldName))
    {
        if(pdRow[strStatusFieldName] != DBNull.Value)
        {
            string sStatus = pdRow[strStatusFieldName].ToString();
            int iStatus = Int32.Parse(sStatus);
            FieldStatus fs = (FieldStatus)iStatus;
            SetStatus2BIField(fs, ref fld);
        }
    }
}

```

```

string param = pathBlaiseInfo+"SheelonU2006"+" /m"+pathBlaiseInfo+
    "SheelonU2006.bwm"+" /NOINITSCREEN"+" /g /x /!" +iLang.ToString()+" /k1"+
    " /e"+pathBlaiseInfo;
if(kLastPressedKey==Keys.F9)
{
    param=param+" /R";
}
broker.FromDsToBdbPath1_2(dsAllData,frCurrentFlat,pathBlaiseInfo+"SheelonU2006.bdb",sPrevDate,sUpdate);
ProcessStartInfo psiB = new ProcessStartInfo(pathDep+"dep.exe",param);

blaise= Process.Start(psiB);
blaise.WaitForInputIdle();
blaise.EnableRaisingEvents = true;
blaise.Exited += new EventHandler(this.BlaiseEnd);
bWait=false;

```

BlaiseIS at Statistics Netherlands

Gerrit de Bolster, Statistics Netherlands

1. Introduction

At Statistics Netherlands the first BlaiseIS questionnaires, created in Blaise version 4.7, were put into production in 2007. In December 2007 a start was made to develop a complete integrated and automated environment for BlaiseIS 4.8 questionnaires. This environment, operational since 1 September 2008, contains a content management system and a dissemination system and supports personalized questionnaires. Additionally a development environment for BlaiseIS 4.8 questionnaires including a questionnaire generator was developed. Questionnaires created with this development environment can be easily exported to the content management system to be put on the web server. As a result at Statistics Netherlands no IT-knowledge is needed anymore to create and deploy BlaiseIS questionnaires. Recently functionality is added to support the creation of download portals in 3 different types.

1.1 Some figures

From 1-9-2008 until 15-2-2009 13 questionnaires for 11 household surveys with a total of 21 reporting periods were put into production. For all these questionnaires 138.685 persons/households were invited to fill in their data. In the same period 2 questionnaires for only 1 enterprise survey with a total of 5 reporting periods were put into production. The total sample for these 5 reporting periods is 30.626 enterprises.

The enterprise survey is compulsory and has a response of about 68%. The persons/households surveys are voluntarily and the response is depending on the size and complexity of the questionnaire. It ranges between 14% and 51%.

2. Design criteria

Although we had already some questionnaires running under Blaise 4.7 it was clearly not the solution we wanted for the future. Blaise 4.7 had too many limitations for that. At Statistics Netherlands the survey departments had a set of demands a system for Internet questionnaires should comply with. Some of them scanned the market for a standard product that supported their demands. None were found. Even they looked into several solutions of other governmental organizations without any success.

2.1 Question types and house-style

First of all it should be possible to apply a wide set of question types. The question types include simple numeric questions as well complicated tables and everything else in between. The texts should be dynamic and a set of fonts, text decoration and colors should be possible. Based upon the experiences with a off-line Blaise/Basil questionnaire for the yearly Production Survey a concept house-style for CAWI questionnaires was created. This house-style included also a predefined set of background colors and a division of the basic screen setup in panels.

2.2 Preview of the questionnaire

Questionnaire developers should be able to see and test their questionnaires before releasing for production. This should be possible without the need of other staff or complicated processes.

2.3 Predefined data

Another important demand is that the Internet questionnaires could be pre-filled with user dependent (personalized) data. This could be T-1 data as well information obtained from other registers or surveys. The data must not be included in the questionnaire itself so the data could be updated without the need of updating the questionnaire.

2.4 Identified access

As a consequence of the previous criteria only the respondent involved should be able to access the copy of the questionnaire with his (confidential) data (pre-filled or not). Therefore a secure login function should be available creating identified access.

2.5 Safe storage of data

Obviously all the confidential data, pre-filled or filled-in by the respondent, should be stored at a safe place where it can not be reached through the Internet by unauthorized persons.

2.6 Low-threshold questionnaire management

Furthermore it should be possible to add or remove questionnaires and reporting periods without the intervention of IT-staff. Especially for short-term statistics this process should be fast (on the fly). Besides, statisticians do have the habit to come up with last minute changes. This flexibility is also needed for the pre-filled data as sometimes samples are created just before sending out the invitations with the access information.

2.7 Regular collection of completed statements

To feed the statistical processes following the collection of the data (like cleansing) on regular bases it should be possible to collect the completed statements from the Internet questionnaires at all times. Therefore the moments this collection process is running should be flexible. Moreover it must be possible at design time to decide if a questionnaire allows correction of already completed statements after sending it. In the first case some kind of version indicator should be present.

2.8 Scalability

As it was hard to say how many Internet questionnaires will be created and therefore how many respondents will be filling in statements the web-server capacity should be scalable without a redesign of the Internet environment.

2.9 Infrastructure

Although this demand was not coming from the statistical departments it is a very important and obvious one. The technology involved should fit in the infrastructure of Statistics Netherlands. This means it should be Windows based.

3. Architecture

With the release of Blaise 4.8 it was possible to for fill all the design criteria described in the previous chapter. Unlike Blaise 4.7 it is based on what is called a server-park which makes it very scalable. The possibility to

create a separate data-server where all the confidential data is stored out of the reach of the users of the World *Wild* Web makes it very safe. A disadvantage is that the data-server itself is not scalable at this moment. A big advantage of using Blaise 4.8 should be that our questionnaire designers are used to work with Blaise. We soon discovered that this was not completely true. I will explain that in the next chapter.

3.1 Basic demands

At Statistics Netherlands electronic data collection as in CASI (self interviewing) is already in use since the early 1990's. With off-line questionnaire tools like CBS-IRIS, EDISENT, EDR and CBSQUEST (all built mainly in Blaise) a lot of primary data was and is collected interactively. Besides that there are more e-channels through which we collect files of all sizes from our respondents as well as other governmental organizations. To manage these flows we have two intertwined applications called MesDesk and QuestManager. These applications are also mainly built in Blaise. Naturally these applications should also support the Internet questionnaires. Therefore they were already extended to support the Blaise 4.7 questionnaires but in a limited way. Installing (and de-installing) the questionnaires was done by staff of the IT-infrastructure department and for collecting the finished statements a process of that same department was used running on predefined moments. For the Blaise 4.8 solution that should change radically. The department responsible for the IT-infrastructure, suffering under severe cut-backs in staff, should only be involved in case a new server should be added to the server-park or a new Blaise version should be installed. All the other processes should be managed by the (non-IT) staff of our data collection department using MesDesk and QuestManager.

3.2 Different Blaise versions

There are currently different Blaise versions in use at Statistics Netherlands. On our production network the default version is still Blaise 4.6. Recently new projects are now beginning to use the Blaise 4.8 version. For the Internet environment Blaise 4.8 is a must. However, on the production network both the applications MesDesk and QuestManager are running under 4.6, but, to interact with that environment from another server, Blaise 4.8 is needed. The IT-architecture group was afraid that introducing Blaise 4.8 on the production network could cause some conflicts, especially if the Blaise services are installed. So it was decided to introduce a separate server, called the Control server, where Blaise 4.8 Developer is installed. Processes that would communicate with the Internet environment should run on that particular server. To do so these processes are written in ASP and IIS is installed on the Control server. The MesDesk and QuestManager applications have full access to a certain folder on the Control server by which they can communicate with it. The ASP's on the Control server are invoked by MesDesk and QuestManager using the GNU-licensed (free) WGET.EXE program. This is some kind of command-line Internet browser.

3.3 Server-parks

Currently the Blaise server-park for our Internet questionnaires consists of a Management server, a Data server and 3 combined Web/Rules servers. On each of them is Windows 2003 Server installed as well as on the Control server mentioned before. For the download portals there will be an additional server-park created using another Data server and another Web/Rules server but the same Management server. Both server-parks (named "onderzoek" and "bestanden") have their own URL's (<https://onderzoek.cbs.nl> and <https://bestanden.cbs.nl>) and therefore their own certificates. The Web/Rules servers are connected to the Internet through a Proxy server. Figure 1 shows this configuration. The bars called DMZ, BackEnd and Production are different networks separated by firewalls.

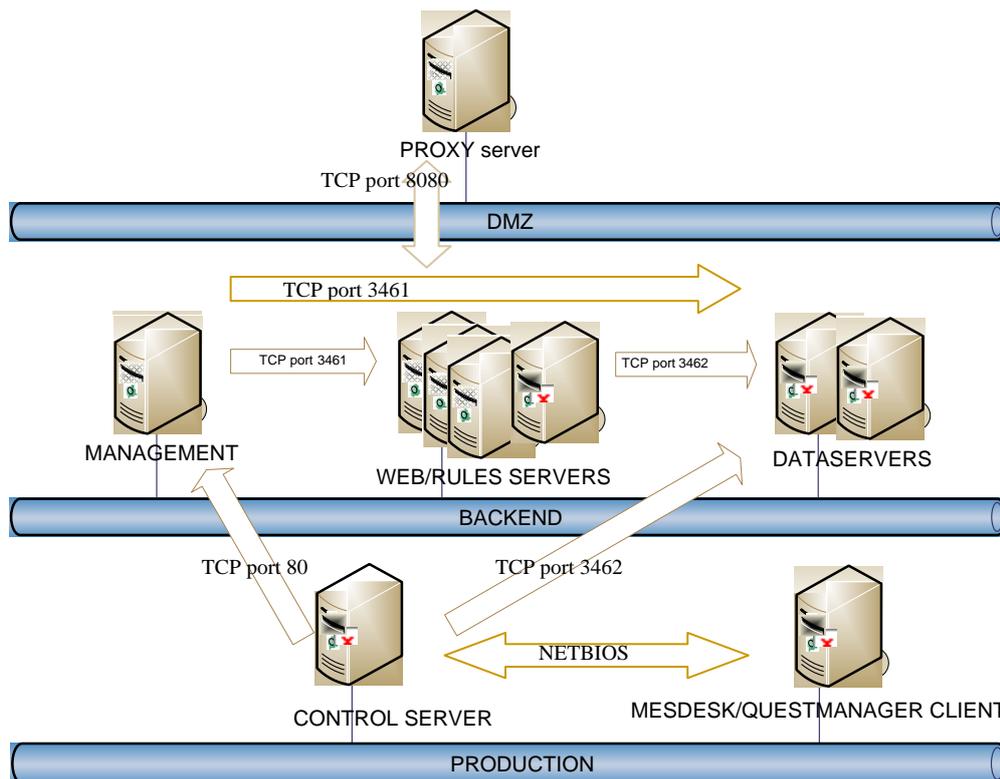


Figure 1.

3.4 Blaise(IS) components

The main processes are written in ASP running Blaise(IS) components for the final actions. This is possible as since version 4.8 all the Blaise Internet programs are in fact components. For interactive use interfaces are added. Without this it would not have been possible to create our BlaiseIS infrastructure. Only the documentation is not very sophisticated. As I am sitting next to the Blaise group in our office this was for me not a real problem.

3.5 Meeting the criteria

Introducing the server-park concept with the separate Data server and the extendible number of Web/Rules servers the criteria about safe storage and scalability was automatically met. Of course, using Blaise we also fit in the infrastructure of Statistics Netherlands. In the next paragraphs the criteria that are met by the chosen architecture are described. The remaining criteria are covered by the development environment as explained in the next chapter.

3.5.1 Identified access

The identified access was obtained by introducing a special questionnaire called "Login". This questionnaire is not called directly by the respondents but invoked through the starter page of every questionnaire. For that we had to change the standard Active Server Page (BiInterviewStarter.asp) that comes with Blaise. Starting an interview the respondent is automatically redirected to the Login questionnaire. This general Login questionnaire searches in the first place if there are more than one periods available for the questionnaire involved. If that is the case the respondent is requested to select the period he wants to fill in. In case only one period is available this period is automatically selected. The information about these periods is stored in a small standard Blaise file that

is available on the Data server with every questionnaire installed. Adding or removing periods this file is automatically updated.

After the selection of the period the respondent is requested to fill in his user number and access code. Using the open HOTP (HMAC-Based One-Time Password) algorithm (RFC 4226) included in a DLL the Login questionnaire checks if the given access code is valid for this questionnaire/period/user combination. This check is programmed as a hard Blaise check. Therefore a respondent cannot pass this part of the Login questionnaire without filling in a valid combination. If he does so, the user number is registered in a log file on the Data server (the "journal") together with a unique session-id supplied by IIS, the browser used, the screen resolution at the respondents' side and the login date and time. After that the respondent is returned to the starter page of the questionnaire. Before starting up the questionnaire the starter page checks if the call was coming from the Login questionnaire. As we even think that these checks are not safe enough we also check in the questionnaire itself if the Login information is registered in the log file at the Login questionnaire (accessing it as an external) to be sure the respondent logged in according to the rules.

Besides the questionnaires we also have open anonymous forms for which a respondent does not have to pass through the Login questionnaire. As soon the data has been filled in the statement is stored and cannot be retrieved by any respondent.

As we do have a general solution for identified access the questionnaire designer is not bothered any more by that issue.

3.5.2 Low-threshold questionnaire management

The questionnaires are installed by the MesDesk application. They are created by the designer using the development environment described in the next chapter. As a result a zip-file is created using an export option and delivered to the department in control of the MesDesk and QuestManager applications. Activating some menu options in MesDesk the questionnaire packed in the zip-file is automatically installed in the right server-park. After that action one more active periods can be added to the questionnaire applying the QuestManager application. To do so a file containing the user numbers has to be imported in the databases of QuestManager. This file must be produced by the statistical department. The corresponding access code are automatically generated by QuestManager and sent to the respondents in a letter. It is possible to divide the group of respondents for one specific period in subgroups. This is done so we can send the invitations out on different moments in time not overloading our reproduction department and the Web/Rules servers.

Using QuestManager a period can be closed for a subgroup of respondents or removed completely. The whole questionnaire can be removed too using MesDesk. If a period is closed and a respondent still selects it the questionnaire will produce a page informing the respondent that he cannot fill in this period for this questionnaire anymore. If the period is removed completely the Login will not be offering this period anymore for selection by the respondent. If it was the only period left the Login questionnaire will inform the respondent that this questionnaire is inactive. After removing the complete questionnaire the respondent gets the standard error (HTTP 404) that the page cannot be found.

The closing and/or removing of the periods are done automatically by registering the closing and removing dates in QuestManager. All the processes can be scheduled so it is even possible to invoke them e.g. every weeknight.

3.5.3 Predefined data

With the QuestManager application it is also possible to transfer predefined data to the Data server. This can be done initially when a period is added to the installed questionnaire or later additionally. This data can consist of different Blaise files used as externals in the questionnaires or can be included in pre-filled statements in the Blaise result file. Of course, this data should be created by the statistical department to which the questionnaire belongs to. The pre-filled statements (with added data from the respondents) for a certain period are removed from the Blaise result file when a period is removed. Of course, the completed statements have then already been collected by the system.

3.5.4 Regular collection of completed statements

Another function of the MesDesk system is to collect the filled-in statements from the Data sever(s). This process checks the server-park(s) registered in MesDesk and retrieves the information about the installed questionnaires from the corresponding Management server. Next all the questionnaires are checked on

completed statements which are copied to the input folder of MesDesk to be transferred further on. The copied statements are not removed but a status field is changed in value so those statements are not retrieved again in the next run. Furthermore the records from the log file are copied giving us information about the browser used, the screen resolution at the respondents' side and the login/logout date and time. The logout date and time is written to the log file when the respondents submits the statement or interrupts the session. The collection process can be scheduled as well or invoked interactively.

The statements are not removed for a specific reason. We do support 2 types of questionnaires related to the times a respondent can send its statement. The type is defined at design-time. In the first type a respondent can send its completed statement only once. If, after sending it, the respondent logs in again he or she is informed that the statement has already been completed and cannot be accessed again. In case of the second type it is possible to open the questionnaire again after sending it. The data can then be edited and submitted again. In the statement a hidden version number is included that is increased automatically. In both cases we need the data to be stored on the Data server to be able to retrieve it after logging in and determine its status or even show the data.

4. Development environment

When we started to develop BlaiseIS 4.8 questionnaires we soon discovered that it was not so easy using the basic Blaise facilities as the BlaiseIS workshop. They are quite technical and Statistics Netherlands is aiming at questionnaire development by specialists not being IT-staff. Another policy of Statistics Netherlands is to apply methodology standards designing the questionnaires. These standards include not only the colors and looks of the questionnaires but also the basic behavior. Besides that the questionnaire designers need to be able to preview the questionnaire under development at any time. They could not use the preview facilities of Blaise caused by the fact that we were using a general Login questionnaire and some specific standard auxiliary files to store information about the questionnaire and its respondents on the Data server. To be able to do that these "externals" must be accessed through Blaise BOI-files that are opened in the rules. The standard Blaise environment is not able to detect that and therefore will not give an accurate preview of the questionnaire. And finally these BOI-files should be created separately and they include Internet host-dependant information.

4.1 BISmenu

For all these reasons we already started creating several auxiliary processes while developing the CAWI environment. Then we grouped them together in a Manipula menu to be able to invoke them easily. The development environment named "BISmenu" was born. To preview the questionnaires IIS was installed on our PC's and BISmenu was extended to be able to install the questionnaires on this so-called "localhost" Web/Rules server. We also included functions to create the necessary auxiliary files and copy them to the "localhost" Data server. The standard configuration that should be used for all the questionnaires (the "house style") was stored in a set of files. From these files emerged the so-called templates.

4.2 Design documents

The example questionnaires we developed to test the CAWI environment were still made by hand programming Blaise sources. In March 2008 it appeared that both the directors from the division for household surveys and the division of surveys for establishments signed contracts already in 2007 with other (semi-)governmental organizations involving large amounts of money to develop and exploit combined Internet questionnaires without being aware that we still did not have the environment to support these kind of questionnaires. The first questionnaire should be operational on September 1, 2008! This discovery put a lot of pressure on the development of the CAWI environment. We contacted the questionnaire designers to get information about these important questionnaires and to inform them about the technical standards that were chosen. The designers for household surveys were using a Word-document for the question texts and types and a Visio-document for the flow to define the questionnaires, until then still being CAPI. Checks and imputations could be found in both documents. Based on these documents their programmers created the Blaise sources by hand adding a lot of design definitions left out in the 2 documents. There were several designer dependant versions of these documents although they used mainly the same principals. As we were involved we tried to create a CAWI

questionnaire from that type of documents. Soon it appeared to be a burdensome task copying and pasting e.g. question texts from the Word-document into the Blaise source. As we are IT-minded the idea of a questionnaire generator using these design documents awoke.

4.3 The questionnaire generator

As I explained in the previous paragraph the first thoughts about a questionnaire generator emerged because of the burdensome copying of questionnaire texts from the Word-document. Because of that the first step was to create a Manipula set-up that created the FIELDS-section of a Blaise source using the Word-document saved as a text file. The next step was aimed at creating the RULES-section. We first examined how we could extract information from a Visio-document. The standard exports just created unusable graphic files. Then Lon suggested that being part of Microsoft-Office it probably could be accessed by an object-model. And so it was (thanks, Lon). Next we examined the Visio-documents created by the questionnaire designers. It appeared that they used it more like a package for drawing (like PowerPoint) than anything else. We soon discovered that Visio could be used in a much better way than that. Finally, in consult with the questionnaire designers, we designed a more standard and structured way how to fill-in a Visio-document as well as a Word-document so we could retrieve through the object model the necessary information to generate Blaise sources for CAWI questionnaires. The questionnaire generator, part of BISmenu, is still growing today as well the functionality of BISmenu itself.

4.4 Templates

Templates play a very important role within BISmenu. They define on a high level the type of questionnaire a designer wants to produce. Although we started with a lot of templates currently we reduced them to a basic set. Several differences were taken out and included in the Word-document as a setting. Thanks to the flexibility of the Blaise components all Blaise (configuration) files (BOI, BMF, BDP, BIP etc.) can be altered without bothering the questionnaire designer with difficult technical oriented interfaces. A template consists of a basic BIS-file (Blaise Internet Specification file), a BLA-file (Blaise source) and several configuration files.

The current set of templates supports the following types of Internet questionnaires:

- O1 – Normal questionnaire with identified access
- A1 – Questionnaire with answers separated from identification
- D1 – Download portal with identified access
- D2 – Download portal with password protected zip-files
- D3 – Public accessible download portal
- F1 – Questionnaire with public access
- M1 – Master/detail questionnaire with identified access (under development)

4.4 Creating a questionnaire

The basic process to create a CAWI questionnaire using BISmenu is as follows:

- Create a Visio- and a Word-document defining the questionnaire.
- Start-up BISmenu.
- Select the menu option “create new questionnaire” typing in the code of the template, the code of the questionnaire (part of the URL), its readable name and (needed in Statistics Netherlands) the size of the user number. The latter is only needed in case of identified access.
- Import the Visio- and Word-document and parser the resulting Blaise source.
- Select the menu-option “update BIS-file”.
- Select the menu-option “add questionnaire” to install the questionnaire on your “localhost”.
- Select the menu-option “period-info”, create one or more new periods (if not yet there) and add the information to the “localhost”.
- Select the menu-option “user-info”, create one or more new users (if not yet there) and add the information to the “localhost”.

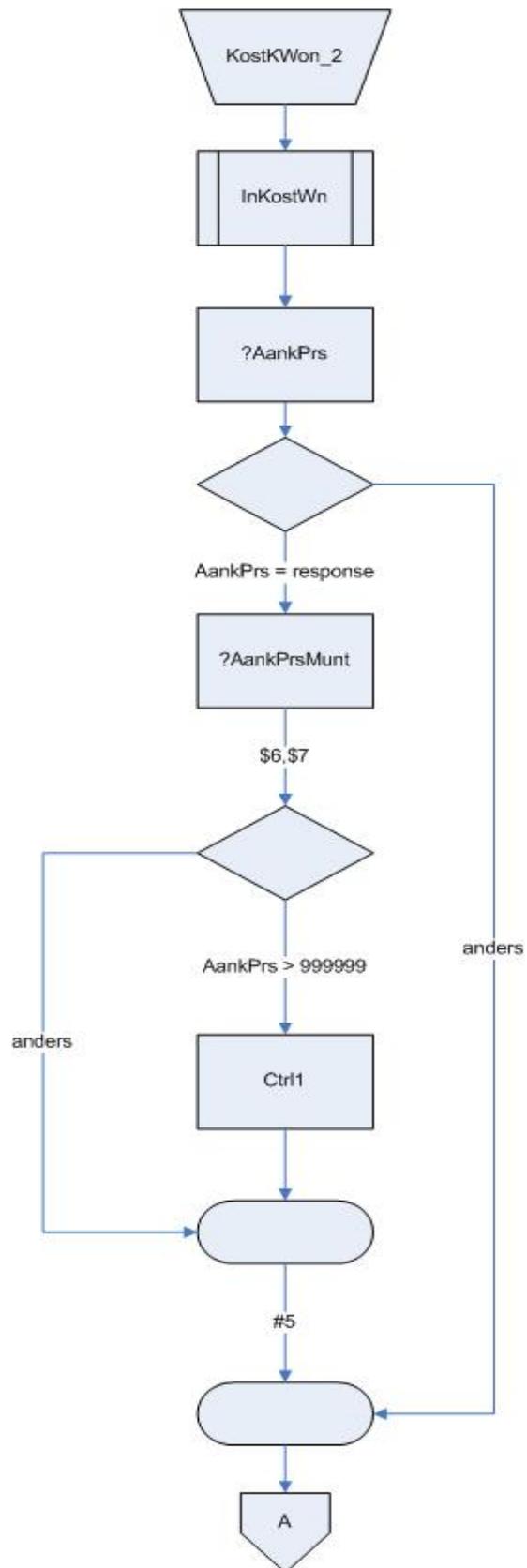
- Optional: select the menu-option “add data” to add pre-filled data to the “localhost”. This data should be created on forehand; some functionality for this is included in BISmenu.
- Start up your Internet browser and check your questionnaire.
- If OK, select the menu-option “export questionnaire” to create a zip-file and send it to the production department.

4.5 The technology of BISmenu

BISmenu is build in Blaise (4.8.1). It consists of a set of Manipula set-ups and VB-script files (Visual Basic Script or VBS) to invoke all the necessary components (Blaise or Microsoft) and uses a set of adapted BlaiseIS files as ASP's (Active Server pages) and XSL's (Extended Style Sheets). To create a similar solution you need in-depth knowledge of Blaise/Manipula and BlaiseIS as well of ASP (which is VBS for IIS) and VBS (both very simple script languages) or bribe the author of this paper.

5. Some samples

5.1 Visio-document



5.2 Word-document

Blok: KostKWon

Condities:

\$1: WonType <> [Boerdery] | [Kantoor]
\$2: WonType = [Boerdery]
\$3: WonType = [Kantoor]
\$4: EenMeerH = [Een]
\$5: EenMeerH = [Meer]
\$6: AankPrsMunt = [Euro]
\$7: (<> \$6) en (AankPrsMunt <> DK|RF)
\$8: RenteMunt = [Euro]
\$9: RenteMunt = [Gulden]
\$10: <> GemVerh(Gemeente,Verh)

Berekeningen:

#1: Trim(WonGem)
#2: Naw.Naam = 'Jan'
#3: Gemeente = Plaats(WonGem).GemcodeI
#4: WonCode = 15
#5: (= \$7): VerkPrys = AankPrs + 1000
#6: (= \$1): WonHulp = N_Hypoth + 10

Controles:

?1: = Plaats(WonGem) "Deze plaats komt niet voor in de zoeklijst."
?2: = GemVerh(Gemeente,Verh) "Deze verhuurder komt niet voor in uw gemeente."
?3: (= \$6) of (AankPrs = response) en (VerkPrys = response) en (AankPrs <= VerkPrys)
"De aankoopprijs mag niet groter zijn dan de verkoopprijs."

Hulpgegevens:

WonHulp
[0..999]

Vragen:

WonGem &RB80H1

In welke plaats <u>woont</u> u?
>>Klik op de knop "Zoeken" om de zoeklijst te openen.<<
STRING[35]=Plaats

WerkGem &RB80H1

In welke plaats <u>werkt</u> u?
>>Klik op de knop "Zoeken" om de zoeklijst te openen.<<
STRING[35]=Plaats

WonType

Type van de woning:
1. Flat, appartement [Flat]
2. Eengezinswoning [Huis]
3. Boerderij [Boerdery]
4. Kantoor aan huis [Kantoor]

Verh &RB80H1

Van welke verhuurder heeft u de woning?
STRING[80]=GemVerh.Verhuurder?GemcodeI=Gemeente

5.3 Screen captures of CAWI test-questionnaires

INV02 - Investerings en Lease - Windows Internet Explorer

Investerings en Lease

Verslagjaar 2008

Direct inzenden aub!

Gerrit de Bolster

Gebruikersnummer:
1234 - 5678

Centraal Bureau voor de Statistiek

Vaste activa in gebruik genomen (in eigendom)

Aankoopwaarde van de materiele vaste activa die in het verslagjaar VOOR HET EERST door uw bedrijf zijn gebruikt én verkregen door koop, huurkoop, vervaardiging in eigen beheer en/of FINANCIAL lease (geen operational lease).

Alle bedragen afronden op 1.000-tallen. Dus € 23 669,75 noteren als: 24.

	Nieuwe activa € x 1 000	Tweede-hands activa € x 1 000	Totaal (A) € x 1 000	Waarvan financial-lease € x 1 000
Grond:	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
Bedrijfsgebouwen:	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
Grond-, water- en wegenbouwkundige werken:	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
Vervoermiddelen:	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
Computers:	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
Machines, installaties en apparaten:	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
Overige materiele vaste activa:	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
Totaal:	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>

TEST4 - Testenquête 4 - Windows Internet Explorer

Testenquête 4

Verslagjaar 2008

Nul

Ikke

Gebruikersnummer:
1234 - 5678

Centraal Bureau voor de Statistiek

In welke plaats woont u?

Klik op de knop "Zoeken" om de zoeklijst te openen.

Weet niet
 Geen antwoord

In welke plaats werkt u?

Klik op de knop "Zoeken" om de zoeklijst te openen.

Weet niet
 Geen antwoord

Type van de woning:

Flat, appartement
 Eengezinswoning
 Boerderij
 Kantoor aan huis
 Weet niet
 Geen antwoord

Van welke verhuurder heeft u de woning?

Weet niet
 Geen antwoord

Wat is uw e-mail adres?

Indien u een e-mail adres bezit vul dat dan a.u.b. hier in.

voortgang:

6. Remarks

Microsoft, Windows, Microsoft-Office, Visio, PowerPoint, Visual Basic (and all the other ones I forgot) are registered ® trademarks of the Microsoft Corporation.

Blaise is a registered ® trademark of Statistics Netherlands.

MesDesk, QuestManager and BISmenu are unregistered names of our applications. No one should even think about it to copy © them!

I want to thank the Blaise team as they helped us a lot to get this CAWI show on the Internet road.

The Use of Metadata in the Design of a Customizable .NET Event History Calendar

Daniel Moshinsky, Mecene Desormice, Seth Benson-Flannery; U.S. Census Bureau, USA

1. Overview

The Survey of Income and Program Participation (SIPP) is a large, long-standing longitudinal survey that collects sources and amounts of income, labor force information, program participation and eligibility data, and general demographic characteristics.

SIPP is undergoing a re-design. As a part of the re-design, it was decided to increase the reference period for the survey from four months to twelve months of the previous calendar year. An Event History Calendar (EHC) is used to help address the concern that the longer reference period may impact respondent recall.

About half of the SIPP questions are asked in the Blaise 4.8 DEP, while the other half are asked in the EHC module, which is written in C# .NET, and invoked from the DEP as a COM object DLL.

This paper focuses on the use of external metadata files and the Blaise 4.8 API to facilitate communication of data and metadata between the Blaise database and the EHC and the role of external metadata in the dynamic generation of a fully customizable Event History Calendar. Advantages, details of implementation, and applications of this approach are discussed.

2. Design and organizational considerations

Several critical survey and organizational considerations guided our selection of a design approach. These considerations are described below.

2.1 Data collection methods

SIPP contains some 40 sections, such as sections relating to employment history, health insurance coverage, assets, utilization of services and demographic history. Not all of SIPP questions are well-suited for being asked in the EHC.

It has been shown that the Event History Calendar method of interviewing is more effective than the standard question list method in improving recall of dates of autobiographical events (Belli et. al., 2001). Therefore, the types of SIPP sections that are asked in the Event History Calendar contain questions involving recall of dates of multiple changes in status throughout the reference period. For example, questions about the dates of job changes or changes in receipt of benefits are collected through the EHC.

On the other hand, questions relating to assets pertain to the reference period as a whole (e.g. "What was the combined value of your assets in the previous calendar year?"), and do not involve recall of dates. Thus, those sections can still be asked in the standard question list method, and can be programmed in Blaise.

2.2 Data exchange

Another requirement is that sections throughout SIPP must be able to exchange data – to pass parameters to each other. For example, EHC sections must be able to use the household roster collected earlier in Blaise, and a commuting module in Blaise must be able to use employment information that had been collected in the EHC. For this reason – and to simplify post-collection data processing – all survey data is stored in a single database.

2.3 Fluid requirements

Since the use of the electronic EHC is new to SIPP and the US Census Bureau, it is understood that requirements may change significantly throughout the development cycle. As the software undergoes internal and field testing,

new questions and sections may be moved in and out of the EHC, wording and composition of individual questions may change, skip patterns may change, and so on. Thus, the design of the EHC must be flexible enough to rapidly accommodate changing requirements.

2.4 Large number of questions in the EHC

There are twenty five sections in the SIPP EHC, each with a different number of potential follow-up questions. For example, once the respondent indicates having been employed from January to September, some forty questions (depending on the skip patterns) will be asked about that spell of employment.

Blaise programmers are accustomed to being able to rapidly program fields and skip patterns -- taking advantage of much behind-the-scenes processing being powered by the Blaise software. Such luxury is obviously not available to programmers working on the EHC in a language other than Blaise – such as a .NET-based language. A lot of code must be written in .NET to correctly place even one question on the screen, to lay out its answer list, to implement an edit check, apply a skip pattern, or to store and retrieve data. With over a hundred fields in the EHC, a more streamlined approach was needed to make the initial programming and subsequent maintenance feasible.

2.5 Re-use of the EHC

Event history calendars are a promising emerging data collection technology. If the integration of the EHC into the SIPP survey is successful, then it is possible that some other surveys may follow suit. The ease with which the EHC can be integrated into a new survey was an important consideration in our design choices.

2.6 Organizational knowledge

Since the majority of surveys developed by our group are programmed in Blaise, there is not a lot of knowledge among the staff of C#. It was important to make it as simple as possible for a Blaise programmer to maintain the EHC module without knowledge of C#, and even to enable such a programmer to customize the EHC for use in a new survey with minimal changes to the underlying C# source code.

In other words, it was important to leave the underlying processing transparent to the survey programmer – much like it is in Blaise.

3. Design Overview

In a nutshell, the SIPP EHC is designed as follows. All sections and all questions that are asked in the EHC are defined as blocks and fields in the Blaise survey datamodel. No fields are hard coded in the C# EHC code. Instead, the C# code includes what can be thought of as templates-processing code. It dynamically generates questions and places them on the screen based on attributes of the metadata – such as question type, question text, answer list text, field length, flow (skip) conditions, and the like.

Three external metafiles – “Questions”, “Topics/Sections”, and “Rules” – are used to drive dynamic generation and placement of fields in the EHC. These metafiles serve as links between the fields defined in the Blaise database (accessed via the API) and the .NET system that displays them.

3.1 Blaise Database

One unified standard Blaise database is used for the entire data collection instrument. All blocks, fields, answer lists, and values that appear in the EHC are contained in the Blaise database and accessed via the Blaise API.

3.2 External metafiles

3.2.1 “Questions” metafile

This metafile describes the metadata needed to display each individual field in the EHC. It includes the block to which the field belongs, the sequential order number in which the field will appear, page number onto which the question will be placed within the EHC (used the same way pages are used in the Blaise modelib), field name and type, length (for text fields), and whether the field allows an Empty value or not.

In addition, this metafile allows a “function” to be attached to a field, and lets the C# system know that additional processing is needed either prior to displaying this field or after a user enters a value. Examples of additional processing are complex skip patterns, edit checks, or dynamic answer lists. Functions are discussed in greater detail below (Section 3.3.2).

The Questions metafile also includes a column to indicate to the system that a help screen is associated with a question. The C# system uses this flag to display the question-specific help screen when the user presses the F1 key while positioned on the question.

TopicID	Questi	Page	FieldName	TypeID	RequiredID	Leng	Last	Function	Help	Column
Job6	0	0	Screene*	Radio Button	Required	1	N			
Job6	45400	0	Header	Title or Header	Optional	12	N			
Job6	45410	0	STARTDAY6	Number	Not Required	50	N	E5.2		
Job6	45420	0	ENDDAY6	Number	Not Required	50	N	E5.3		
Job6	45430	0	NUMBER_OF_JOBS	Number	Not Required	50	N	E15.4=V		
Job6	45440	0	JBORSE6	Radio Button	Required	1	N			
Job6	45450	1	WRKTYPE6	Radio Button	Required	1	N	E15.5=K		
Job6	45460	1	KNDWK6	String	Not Required	150	N			
Job6	45470	1	ACTVT6	String	Not Required	150	N			
Job6	45480	2	EARN6		Not Required	100	N	E3.1=h-1		
Job6	45485	2	TAKEHOME6	Radio Button	Required	1	N			

Figure 3.2.1 The “Questions” metafile

3.2.2 “Rules” metafile

Another metafile is used to describe the rules – i.e. the conditions for when a question is brought on-path or off-path based on answers to preceding questions. This file allows for implementation of simple conditional statements without writing any code in C# -- the person entering the rules must only enter the question ID of the antecedent question and its value (i.e. “if question 30 is Yes”), as well as the question ID of the consequent and its on- or off-path status (i.e. “then question 31 is Off-Path”).

In the screenshot below (Figure 3.2.2), the antecedent is described in column FieldName, and the consequent is described in column GoToField. A value of 0 in the Action column indicates that the GoToField is off-path for the given response to FieldName, and a value of 1 indicates that it is on-path. In the screenshot below, if values of 19, 20, or 21 are entered in field GRADE, then the GRADEREP field is off-path, but field FTPT is on-path.

The “Compound” column allows for an “AND” combination of more than one value to determine status of the GoToField.

This metafile handles the majority of skip patterns in a questionnaire. Some questions, however, require a more complex skip pattern than a simple AND/OR conditional. For instance, values from outside the block may be

considered, or values may be combined in a complex way. Handling of more complex rules like these is done through Functions (see section 3.3.2 below).

Topic	FieldName	Response	GoToField	Action	Compound
Enrollment	Grade	19	GRADEREP	0	
Enrollment	Grade	19	FTPT	1	
Enrollment	Grade	20	GRADEREP	0	
Enrollment	Grade	20	FTPT	1	
Enrollment	Grade	21	GRADEREP	0	
Enrollment	Grade	21	FTPT	1	
Job	JBORSE	0	STARTYEAR	1	
Job	JBORSE	0	STARTMON1	1	
Job	JBORSE	0	STARTDAY	1	
Job	JBORSE	0	JOBEND	1	
Job	JBORSE	0	ENDDAY	1	
Job	JBORSE	0	RSEND	1	
Job	JBORSE	0	RENDB	0	

Figure 3.2.2 The “Rules” metafile

3.2.3 “Topics/Sections” metafile

This file describes the metadata of the instrument at the section level. It consists of a table of topic IDs – that determines the order in which topics are displayed in the EHC – and topic names corresponding to block names in the Blaise datamodel. An additional column describes the maximum number of spells each topic can have, as requested by the specifications (due to the great number of blocks in this person-based survey and the size limitations of the Blaise database, it was necessary to limit the maximum number of spells). For example, no more than 5 changes of residence during the reference period may be recorded for each household member. If the user attempts to enter a 6th residence, an edit check appears.

Additional topic-level attributes (perhaps a topic-level help screen or edit check) may subsequently be included in this table, if needed.

TopicID	Topic	BlockName	MaxSpells
1	Landmarks	Blandmark	
2	Residences	Bresidency	5
3	Marital History	BMARITAL_HISTORY	6
4	Mom	Bmom	6
5	Dad	Bdad	6
6	Enrollment	BEduc_Enrollment	6
7	Job	BJob1	4
12	Job6	BJob6	4
13	NoJob	BNoJob	4
14	SSI	BSSI	3
15	FS	BFS	3
16	TANF	BTANF	3
17	GenAssis	BGenAssis	3
18	WIC	BWIC	3
19	Private1	BPrivate1	3
21	Medicare	Bmedicare	3
22	Medicaid	Bmedicaid	3
23	Military	Bmilitary	3
24	OtherCoverage	BOtherCoverage	3
25	NoCoverage	BNoCoverage	3

Figure 3.2.3 The “Topics/Sections” metafile

3.3 The C# system

3.3.1 Dynamic processing

All sections and fields are generated and displayed at run-time by querying the Blaise API and the metafiles described above. The C# code contains implementations of generic methods for each operation necessary to display blocks, fields, and answer lists of every type (e.g., string, numeric, radiobutton, checkbox, dropdown), as well as methods to process the rules, display edit checks, retrieve and store data, and so on.

3.3.2 Functions

More complex event processing is handled through “functions” -- methods written in C#. The “Questions” metafile described in Section 3.2.1 contains a column with a function number (e.g., “E5.2”) that associates one or more methods with a particular field. When a field loads, or when a value is entered in it, the system “sees” that the field has an associated function (or functions), and the system proceeds to process the instructions in that function.

A function may perform one of several jobs. It may be called upon to process an edit check, determine a skip pattern, populate a dynamic answer list, or perform some other specialized role.

The skip patterns relegated to the functions are ones that are too complex for the Rules metafile to handle. Still, the programming involved in writing a new function is fairly straightforward and requires very little knowledge of the C# language beyond familiarity with the syntax of writing conditional statements.

Most edit check processing functions are generic and shared by many fields. For example, the range check function – which checks to see that a numeric value falls within a specified range – is used by nearly every numeric field in the EHC. The re-use of functions across fields is accomplished by different parameters being associated with different fields. Parameters include whether the check is hard or soft, and may include ranges of valid values, and so on.

For example, the field EARN6 (a question about earnings) in Figure 3.2.1 has a function associated with it. The full value in the Functions column for that row reads “E3.1=h-1000-999999”. The letter “E” in front of the function number indicates that the function will execute following an Entry in that field. The number of the

function is followed by the letter 'h' – indicating an instruction to display a hard check if a value is entered that is either less than 1,000 or more than 999,999).

Once the generic edit check and skip pattern functions have been programmed, they can be re-used by many other fields in the instrument simply by passing a different set of input parameters to the functions. As a result, all numeric fields can share the same range check function, and skip pattern functions can be shared by passing names of involved fields as parameters.

The functions feature of the design allows for sophisticated error and flow handling within the EHC, while keeping the EHC easy to maintain and customize for Blaise programmers unfamiliar with the advanced features of C#.

4. MS Access utility

During initial development, the metafiles were being maintained as comma-delimited text files and modified either directly with a text editor, or via Excel. As development continued, and the number of questions implemented in the EHC grew, so did the size of the text files, and it became difficult to maintain the files manually.

In order to build a more reliable and maintainable application, we then developed a small MS Access database application to organize the information. This tool provides a way to ensure consistency of the data entered by performing some basic checks on the validity of entries and using dropdown lists to restrict possible values. It allows us to maintain, update, and modify the metadata more accurately.

5. Examples of metafile use

5.1 Adding a question

To add a question, one would first add a field to the correct block in Blaise, and then simply add a line to the Questions metafile table, filling all required cells.

5.2 Re-ordering questions

To change the order in which questions come up (e.g., to reverse the order in which State and County are asked), one would simply modify the QuestionNumber column in the Questions metafile.

5.3 Changing question types

To modify the type of the question (e.g., change a numeric field to a text field), one would simply change the value of the Type column in the Questions metafile.

5.4 Changing question or answer list wording

In order to change the wording of a question or an answer category, one would modify them in the Blaise datamodel and rebuild the Blaise project. The EHC would pick up the new wording because the EHC retrieves the wording from the Blaise datamodel via the API.

5.5 Adding a new EHC section

In order to add a new section to the EHC, first a new block needs to be defined in Blaise. Then, a line must be added to the Topics/Sections metafile. Subsequently, questions and rules (if any) must be entered in the other two metafiles.

6. Conclusion

The three-level interaction among the EHC .NET code, the Blaise API, and the external metafiles yields a robust data collection system that not only includes all the features of an Event History Calendar, but also nearly the same question flow and error checking that our users have come to expect from Blaise surveys.

Almost all C# processing is transparent to survey programmers who need to only concern themselves with entering the metadata into Blaise and into the external metafiles, and perhaps also with modifying a few of the functions.

Specifying EHC fields in the metafiles and the Blaise database has the additional advantage that enhancements to the EHC functionality and layout can be carried out independently of decisions about specific items. Fields and rules can be modified in the metadata without affecting the datamodel – the C# DLL does not need to be rebuilt in order to effect the changes to the metafiles. This makes maintenance after deployment easier, and allows for an immediate fix to certain kinds of problems that might occur in the field.

The SIPP EHC system is still young and in continuous development. Many technical challenges have been overcome, and quite a few still remain. All of the core design features discussed above have been successfully implemented, but we hope to improve upon some of them in the future. For example, it would be particularly helpful to validate metadata entry and make it more user-friendly. Another possible enhancement is to import parts of the metafiles directly from specifications.

Lastly, we have encountered multiple other challenges while developing the EHC, such as issues with invoking the COM object DLL from Blaise, exchanging values through the Blaise API, cleaning off-path data in the EHC, and others. The description of those challenges lies outside the narrow scope of this paper, but we hope to report on them in the future.

7. References

Belli, Robert F.; Shay, William L., and Stafford, Frank P. . Event History Calendar and Question List Survey Interviewing Methods: A Direct Comparison. *Public Opinion Quarterly*. 2001; 65(1):45-74

Development and Programming of an Employment Event History Calendar in the Panel Study of Income Dynamics

April Beaulé, Mary Dascola and Youhong Liu, University of Michigan

1. Introduction

The Panel Study of Income Dynamics (PSID) is a nationally representative longitudinal study of approximately 8700 U.S. families. Since 2003, the PSID has used various Event History Calendars (EHC's) for capturing retrospective reports. The first electronic version of the EHC was built as a stand-alone component to collect employment history of the main respondent and their spouse. In 2007, new childhood health calendar was added. The new calendar was designed to collect information on the most common childhood medical conditions including asthma, diabetes and allergies. This new childhood health calendar was programmed using Visual Basic but integrated into the Blaise application so it appeared seamless to the interviewer. The Blaise DEP (Data Entry Program) called the VB application and data from the childhood calendar was written back to the bdb using a Dynamic Link Library (DLL). Given the success of the childhood health EHC, in 2009 the complex employment Event History Calendar was reprogrammed in a similar integrated fashion.

This paper will discuss the development of the new employment EHC, specifically how we incorporated a number of checks and warnings to the interviewers in order to meet the complex requirements of collecting information about employment and labor force participation.

2. Background

The Panel of Income Dynamics had been on an annual interviewing cycle from 1968 through 1997. Due to budget constraints, the study then went to an every other year interviewing cycle. Interest in the Event History Calendar (EHC) data collection peaked during this transition period, as there was concern about whether respondents would be able to accurately answer questions about moves and especially employment spells over the two calendar years prior to the interviewing year. Building on the on the retrospective memory work of Robert Belli, Ph.D., a cognitive psychologist, the PSID conducted an experiment in 1998 on the use of an EHC to collect data in the core domains used in the PSID including housing moves and employment spells. The experiment showed that the EHC method of interviewing provided more accurate autobiographical retrospective reporting in comparison with the standard question list method.¹

A main focus of the PSID survey is employment, income, housing and wealth. Since spells of employment and unemployment are central to the PSID, that section was re-programmed in 2002 as an EHC instead of a standard question list. When the PSID was reprogrammed in 2002, Blaise offered no method to present a calendar type grid. Therefore, the employment calendar was programmed using Visual Basic with a Microsoft Access database on the back end for data storage. Due to lack of stability issues that we faced with our first attempts using DLL's, the decision was made to keep the stand-alone Access database for storage. The University of Michigan programming staff developed an in-house interface that moved data from Blaise to VB and from VB to Blaise again. The result was that the PSID application was broken into five separate applications and various data sources.²

¹ From PSID website see overview of Calendar Methods Study
<http://psidonline.isr.umich.edu/Data/documentation/ehc/PSIDcalendarMethodsStudy.html>

² Beaulé, A. Leissou E. and Liu. (2007): Experience Using and Event History Calendar in the Panel Study of Income Dynamics, Proceedings of the 11th International Blaise Users Conference, Maryland, United States, September 2007.

3. Prior Experience with EHC Data Collection

The EHC provided the interviewer with a user-friendly interface. Data collection feedback from the field indicated that anecdotally, it did indeed help respondents with recall. The EHC provided the an interviewing advantage of the free flowing format where the both the respondent and the interviewer could move from domain to domain and time period to time period in using a semi-structured format that aided respondent recall. The downside to the unstructured format was a sharp decline in data integrity.

The concept of labor force participation is central to the PSID so that the notions of vacation time, out of the labor force spells, and employment spells must all correspond with each other and all weeks of the year must be accounted for. For example, it is important to know the exact number of weeks worked for a given individual. Elapsed work weeks cover the period from when the employment spell began to when it ended. The number of 'weeks worked' is those elapsed weeks minus 'time away' which can include sick time and vacation time for example. Periods where no employment spells exist are also important and we refer to these periods as time 'out of the labor force' or 'Not Working.' For those times when respondents are out of the labor force, we also distinguish between times when they are looking for employment and when they were not looking for employment.

In the earliest version of the employment calendar, these types of employment or out of the labor force spells were collected but we were unable to provide the hard and soft-consistency checks to the interviewer so they could make on the spot corrections with help from the respondent. The application allowed overlaps between various types of spells and each of those inconsistencies had to be resolved by the data processing staff in order to determine overall weeks worked and time spent out of the labor force.

Each type of spell was collected on a separate tab which made it difficult for interviewers to really see how each of the domains corresponded with each other. There was an attempt to provide the interviewer with an overall timeline at the top of the screen just below the question text, but in order for the interviewer to see the details for any of the spells he or she would have to move to each tab to review those. (Figure 1)

Figure 1: Employment Spells—Early version EHC

 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	MAR

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	MAR

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

OK Cancel Delete

In addition to the internal spell consistency problems, we also faced consistency problems with data collected earlier in the Blaise question list portion of the interview. One of our key variables is current employment status. This variable is a multiple choice question that is asked prior to entry into the calendar. For people who say their employment status is 'working now', they must have current main job in the employment domain in the calendar. Conversely, if the interviewer lists a current main job but listed the person's employment status as retired for example, then either the person's employment status should include 'working now' or the job listed should not be a current job. These types of inconsistencies are best resolved on the spot during the interview.

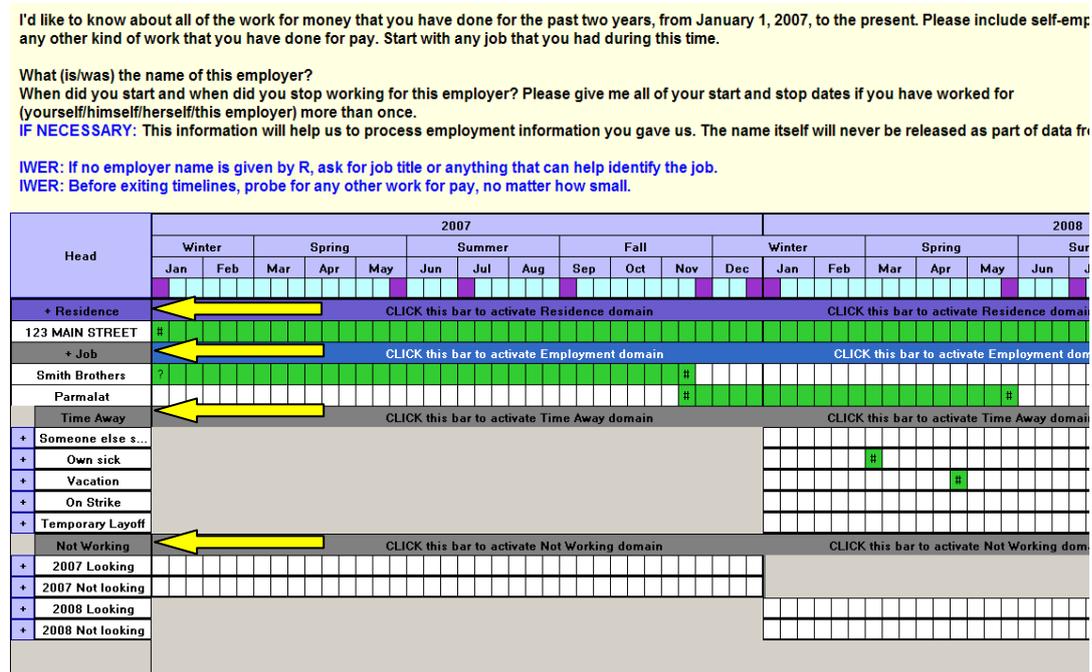
4. Goals for the New Employment EHC

The main goals for of the new version of the employment EHC were to 1) provide a visual guide to the interviewer of the domains and how they closely relate to each other 2) provide the interviewer with checks to help resolve problems during the interview. The new EHC forms were programmed in C#.net.

Some domains need to overlap; spells of 'time away' need to correspond to spells of employment. Some domains should not overlap; spells of 'out of the labor force' should not overlap with spells of employment. The new EHC also needed to provide a way for us to provide consistency checks with the key variable 'employment status' which is located outside the calendar itself. These consistency checks should be easy for the interviewer

to interpret, to respond to, and assist them in soliciting information from the respondent during the interview to resolve data conflicts.

Figure 2: New Version of Employment EHC

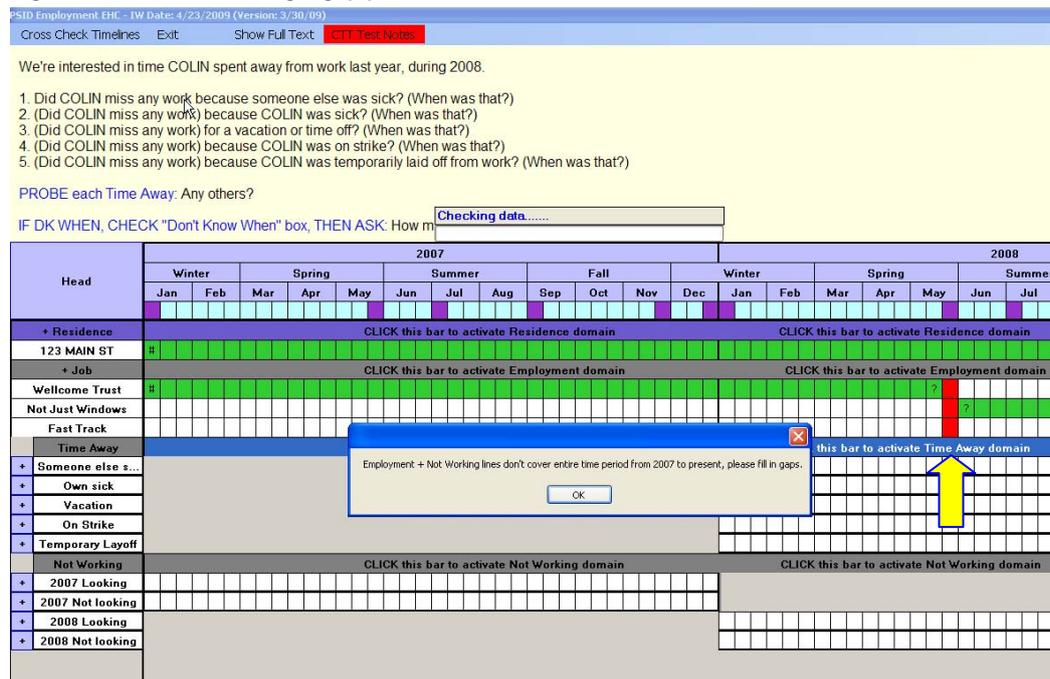


In the newer version of the EHC (Figure 2), you can see that instead of having the domains listed on separate tabs, they are listed all on one screen with the domains listed along the left hand side. This allowed the interviewer to see the rows for each of the domains at the same time and to see where gaps and overlaps are happening.

Since each cell represented 1/3 of a month—we had to allow for some overlap between different types of spells such as employment and unemployment but wanted to warn the interviewer when we had gaps or overlaps occurring. We used several techniques to alert the interviewer in a visual way about the overall integrity of the employment data.

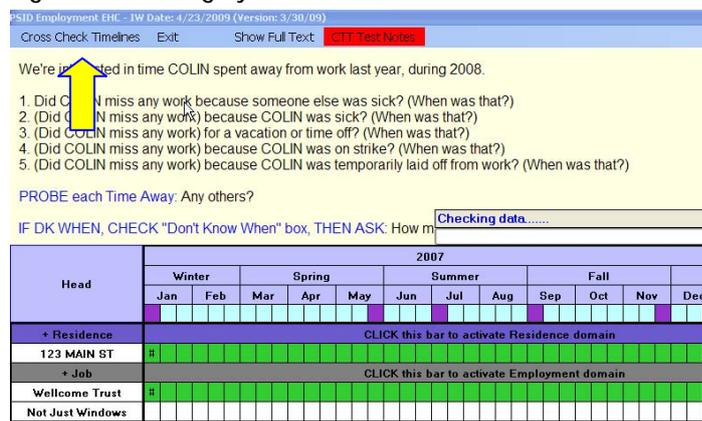
First, we used color to show the interviewer where problems exist. We took our inspiration from traffic signs; green to indicate these portions of the time lines are consistent and red to indicate that a data conflict should be resolved before the interview can move forward. In Figure 3, the interviewer has initiated the data integrity check and the feedback he or she receives is that there are cells, highlighted in red in the last third of May, which are not accounted for. All cells should be accounted for by a period of employment or a period of 'out of the labor force.' In the instrument we have labeled the types of out of the labor force as 'Not Working.' In addition to the problem cells marked in red, the interviewer gets a warning; 'Employment + Not Working lines don't cover the time period from 2007 to present, please fill in gaps.' There are two possible solutions, either the jobs should be concurrent or the interviewer has to mark that time as some type of 'Not Working.'

Figure 3: Checks indicating a gap problem



Since the data entry task in the calendar is semi-structured, we wanted the checks and warnings to be as unobtrusive as possible. The goal was to allow the interviewer free navigation between domains to collect all the information in whatever order without being tripped up by constant real-time rule checking. Our solution was to provide them with a menu option (Figure 4, below) that would initialize the checking once the interviewer felt all the information had been collected and was complete. The interviewer could then make adjustments based on the consistency check feedback and re-check as necessary until all the conflicts were resolved.

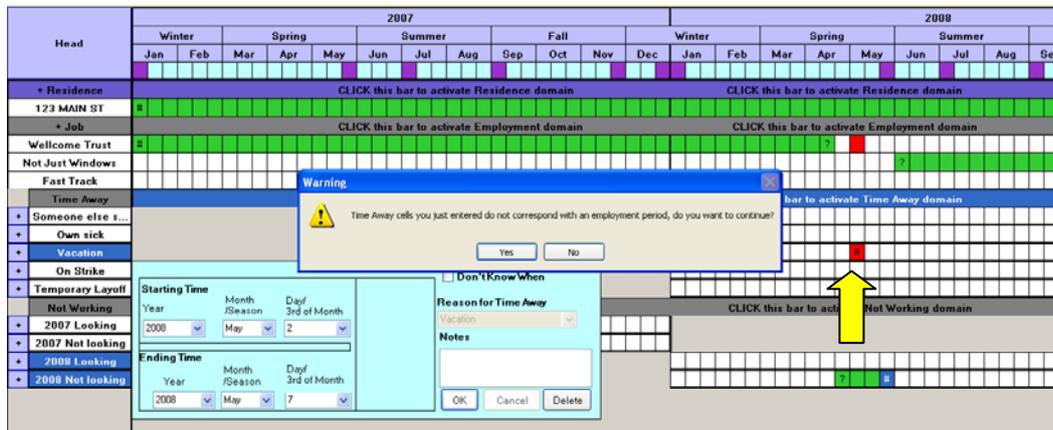
Figure 4: Data Integrity Check Button-'Cross Check Timelines'



For example, a common mistake made in data entry is what we mean by vacation. When respondents think of vacation, they think in terms of the time they went to the beach with the family. When we define vacation, we are talking about times when you have a spell of employment but you are not working because you are on vacation. Respondents will report times of vacation when they are not employed—that time is not considered vacation to us, this is time 'out of the labor force.' In order to help the interviewer understand this difference, we train them about concepts and then we provide consistency checks that remind them that we either have to have a spell of employment to go along with the vacation or not to consider that time vacation but rather, it should be coded as 'out of the labor force' otherwise known as 'Not Working' (Figure 5).

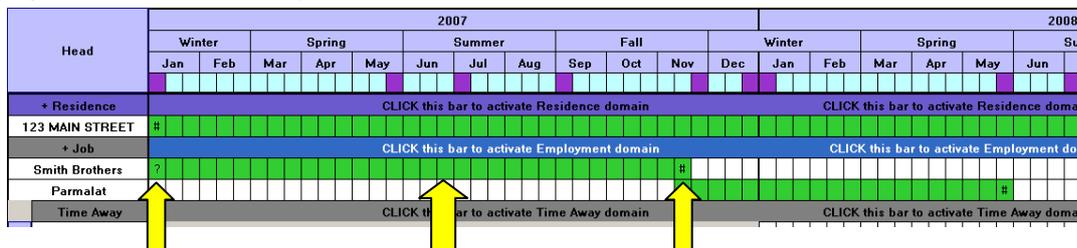
In this scenario, the interviewer has gone to the 'Time Away' Domain and tried to enter vacation for a period of time when the Respondent has no employment cell. The consistency check warns the interviewer 'The Time Away cells you just entered do not correspond with an employment period, do you want to continue?'

Figure 5: Incorrect Reporting of 'Time Away' – Vacation Spell



To further enhance the visual feedback about the timelines, we used symbols at the start and end of each time line and across strings of cells to let the interviewer know whether the timeline had a definite start or end date (actual day, month and year) or whether it was slightly fuzzy (first third of the month), or whether we knew some information but it was less specific. For example, this time line below (Figure 6) shows that the employment spell for 'Smith Brothers' has a fuzzy start date but a definite end date. We used symbols '?' in the first cell to indicate a fuzzy start and '#' to indicate a definite end date. For spells that are much less specific such as two weeks of vacation taken in the year 2008 but the respondent doesn't know when, we fill the cells on that timeline with '?'.

Figure 6: Exact versus Fuzzy Start or End Dates



During PSID post-processing and editing, we spend many hours cleaning up cases where our respondents have overlapping jobs and time away spells. In order to confirm the elapsed 'work weeks', we added a new calculation and routine in 2009. Each cell represents one third of a month. Across the length of the year, each cell represents approximately 1.45 weeks. For respondents with overlapping job spells, we wanted to confirm with them the number of weeks we think they were employed after the employment calendar is completed.

Here is an example of a person with three distinct jobs. Job 1 begins in the second 1/3rd of May and runs through the second 1/3rd of June. Job 2 is from the first 1/3rd of June through the second 3rd of June. The 3rd job runs from the first 1/3rd of June through the last 1/3rd of June. The cells in blue indicate that these cells overlap with cells from another job. The cells in pink indicate that these cells do not overlap with cells from another job.

Figure 7: Calculating and Confirming Elapsed Work Weeks

	May	June	July	TOTWSKJob
--	-----	------	------	-----------

	1	2	3	1	2	3	1	2	3	
Job 1										$4 * 1.45 = 5.8$
Job 2										$5 * 1.45 = 7.25$
Job 3										$3 * 1.45 = 4.35$

Summary Indicators

WTROverlap	NumConcur	NumUnique	Numthirds	Numwks	Rounded
1	3 (Blue)	4 (Pink)	7	$7 * 1.45 = 10.15$	10

Behind the scenes, we process a few items: first we count the number of cells for each job and we multiply them by 1.45 which is the average number of weeks that 1/3rd of a month represents. Then we calculate the maximum number of concurrent cells and the number of unique cells across all jobs and the total is the number of elapsed 1/3rds of months. In this example we have 7 elapsed thirds of months * 1.45 gives us 10.15 weeks, we round that down to 10 weeks and then we confirm this with the respondent.

Q1. : From the information you have given me on your all of your jobs, you worked approximately 10 weeks in 2008, does this sound correct?

The respondent has a chance to let us know if this is a correct reflection of their elapsed ‘work weeks’ and if it is not, they can give us an improved estimate. This confirmation will significantly cut down on hand corrections for ‘work weeks’ calculations during the processing phase.

5. Training Interviewers on the use of an EHC

Interviewer training for the EHC focused on familiarizing interviewers with the new single-screen setup in terms of navigating domains and entering timeline data, and reconciling timeline errors. Data items were entered by choosing a starting cell and then an ending cell. Depending on the item, pop up windows open (Figure 8) so additional information such as employer name can be entered.

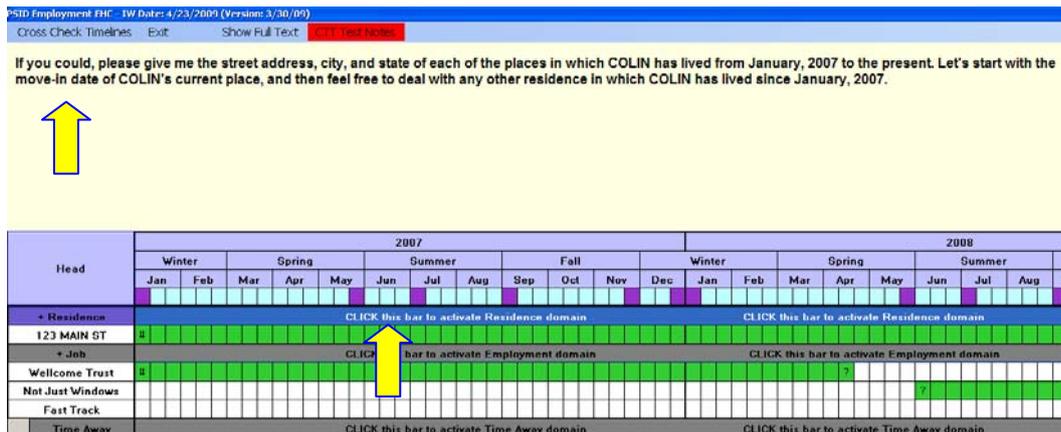
Figure 8: Entering Data Items

The screenshot shows a software interface for entering employment data. On the left is a vertical list of domain categories: Wellcome Trust, Not Just Windows, Fast Track, Time Away, Someone else s..., Own sick, Vacation, On Strike, Temporary Layoff, Not Working, 2007 Looking, 2007 Not looking, 2008 Looking, and 2008 Not looking. The main area features a calendar grid for the year 2008, with columns for 'First third', 'Mid third', and 'Last third' of the month. A data entry form is overlaid on the calendar, containing fields for 'Starting Time' (Year: 2007, Month/Season: Jan, Day/3rd of Month: 1) and 'Ending Time' (Year: 2008, Month/Season: Apr, Day/3rd of Month: 1). There is also a 'Current' checkbox and an 'Employer' dropdown menu set to 'Wellcome Trust'. A 'Notes' field is present below the employer dropdown.

Since all domains and timelines are now on one screen, not all information is available for viewing at once. In order to view timelines that ‘fall off’ the screen, interviewers were trained on general navigation techniques such as scrolling and dragging, techniques not needed in the previous EHC. For instance, the two and a half year time period that we are concerned with does not fit entirely on the screen, and interviewers must use horizontal scrolling to get to the latter part of the time period. Depending on the number of timelines, not all can be viewed at one time. In this case, interviewers must use vertical scrolling to get to those timelines towards the bottom of the lower pane. Finally, interviewers were shown how to relocate a blue data entry box by clicking and dragging, in order to better view the box itself or the timeline behind it.

In order to access the question text and timelines for a specific domain, interviewers were trained how to activate a domain by clicking on its labeled ‘domain bar’.

Figure 9 : Using the Domain Bars to Navigate



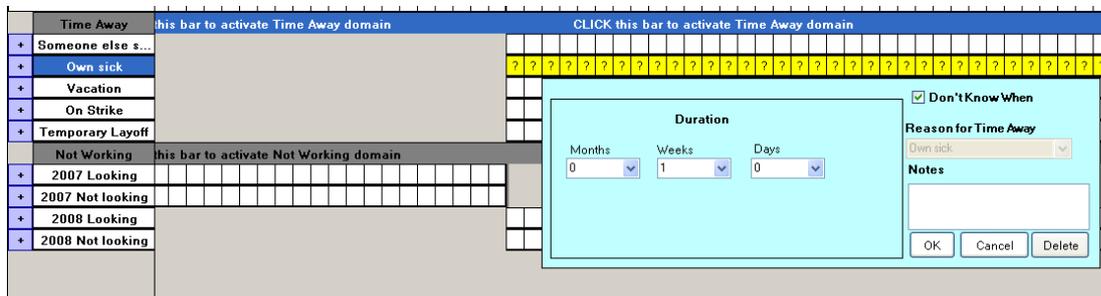
Clicking the domain bar controls presentation of corresponding question text in the top pane and timeline data entry in the bottom pane – neither question text nor data entry for a domain can be accessed without first clicking the domain bar (Figure 9). The interviewer can see which domain is active as the active ‘domain bar’ is highlighted in blue. On the other hand, it is not necessary to activate a domain in order to edit a completed timeline in the domain – that can be done by simply clicking on the title of a timeline regardless of which domain is currently activated.

We relied on a greater amount of hands-on practice during training, in relation to lecture, in order to provide interviewers adequate time to become skilled at moving around the EHC, pointing and clicking to activate domains, and entering timeline data. Additional one-on-one assistance proved to be of great value for many interviewers.

A critical addition to the new EHC is the Cross Check Timelines menu option, which checks for inconsistencies in timelines within and across domains. Inconsistencies are varied and complex. For instance, Time Away timelines are allowed only where there are corresponding Employment timelines otherwise, the Time Away spell is considered invalid. Conversely, Not Working timelines are allowed only where there are **no** Employment timelines. Time Away timelines cannot overlap each other (one can be on vacation or sick, but not both at the same time), nor can Not Working timelines overlap each other (a person is either looking for a job or not looking, but not both at the same time). Timelines may overlap each other by just one cell (third of a month). An exception is an overlap where any start or end date is [DK]. An exception to the exception is a Time Away cell that falls within a period of No Employment, which is never valid, even if a start or end date is [DK].

Below is an example (Figure 10) of entering a period of sick time for the previous year. The respondent knows they had a week of sick time in 2008 but the do not know when. The timeline for this row is marked in yellow with ‘?’s. In the database, we know that we have a period of own sick time in 2008 and the duration is one week.

Figure 10: Entering a period of ‘Sick Time’ for last year ‘Don’t Know When’



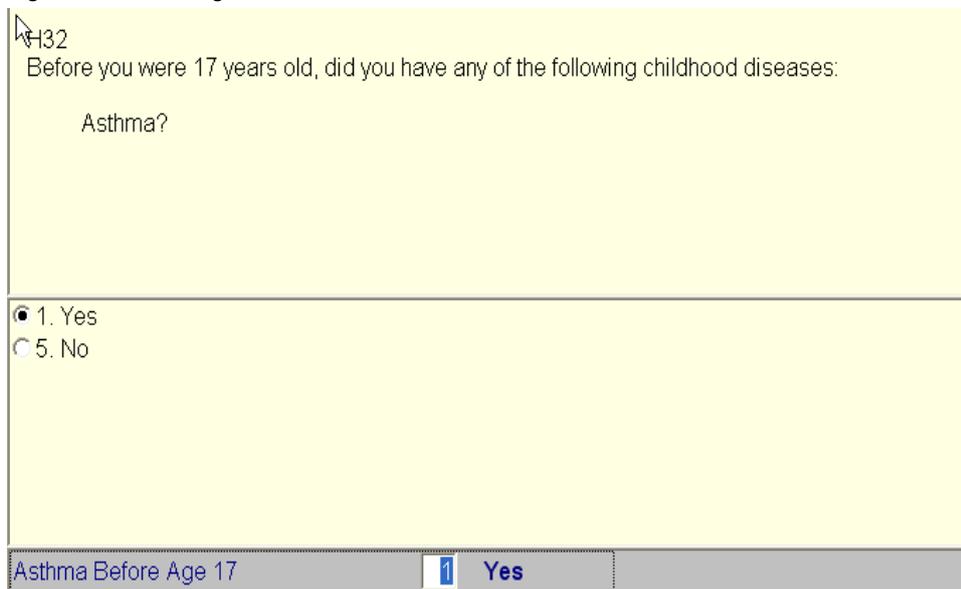
Because of the number and complexity of errors, training emphasized the use of Cross Check Timelines, rather than memorization of all possible inconsistencies and what order the program checked for these. Interviewers were trained to use Cross Check Timelines frequently during the process of moving through the domains, rather than waiting until all domains were completed and all timelines entered. It was recommended to fix an error, by

verifying with the respondent, as soon as the error was presented, and then to run Cross Check Timelines immediately to make sure that the error was indeed resolved.

6. Experimenting with EHC using Dynamic Link Libraries

In 2007 we incorporated the childhood health calendar into the application. The questions focused on childhood health conditions. Since the reference period focused on the respondent's early life, the calendar method of data collection was chosen. Instead of having a secondary data storage file, 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. When conditions were endorsed in the screening section (Figure 11) then control would pass from Blaise to the EHC where follow up questions about when the respondent had the condition were asked. If no conditions were endorsed by the respondent, Blaise would skip the calendar subroutine and continue on to the next field on route.

Figure 11: Screening Question in Blaise for the EHC Childhood Health Calendar



The screenshot shows a Blaise interface with a yellow background. At the top left, there is a mouse cursor and the number '132'. The main text reads: 'Before you were 17 years old, did you have any of the following childhood diseases:'. Below this, the word 'Asthma?' is displayed. At the bottom of the question area, there are two radio button options: '1. Yes' (which is selected) and '5. No'. At the very bottom of the screenshot, there is a grey bar containing the text 'Asthma Before Age 17' on the left, a small icon in the center, and the word 'Yes' on the right.

The data collected in the calendar was then sent back to the Blaise bdb via the DLL. The use of the DLL for handling data movement and storage was extremely successful. We had no cases with missing or anomalous calendar data.

7. Programming Logistics

The employment EHC DLL is written in C# .net 2.0. In addition, BCP 4.8 is utilized. The first step in programming the EHC is to create data structures in Blaise that work with all the domains in the external application.

- Bemploy (Figure 12) defines employment time line data. It stores start and end date, employer name and a few other pieces related information. The field xXML stores all other data in the block for quick data saving and retrieving in a C# form. xDisplay and sSpanDisplay are constructed in the C# program for later text displays in Blaise fields. Array [1..10] of Bemploy indicates that we are allowing a maximum of 10 employment time lines
- BtimeAway(Figure 13) has the similar data structure as Bemploy. Five arrays are defined to represent 5 time away reasons: Someone Else Sick, Own sick, Vacation, On Strike and Temporary Layoff.
- BnotWork behaves similar as Btimeaway.

Figure 12: Employment Blaise Block Definition

```
BLOCK BEmploy
  Fields
    AQSN : TAQSN
    sXML /"Xml string for easy alien router programming":string[2000]
    sDisplay /"display for summery screen" : string[1000] {part of xXML}
    sSpanDisplay /"display for summery screen" : string[1000] {part of xXML}
    sYear /"Start Year " : 1901..9997,DK,EMPTY
    eYear /"End Year " : 1901..9997,DK,EMPTY
    sMonthSeason /"Start Month/Season (start month)" : 1..24,DK,EMPTY
    eMonthSeason /"End Month/Season (End month)" : 1..24,DK,EMPTY

    sDay / "Start Day (start day of the event)": 1..43,DK,EMPTY
    eDay / "End Day (End day of the event)": 1..43,DK,EMPTY
    Employer / "Event Description": string[100],DK,EMPTY

    Notes / "Event Notes": string[200],DK,EMPTY
ENDBLOCK

EmploySection : Array [1..10] OF BEmploy
```

Figure 13: Time Away Blaise Block Definition

```
BLOCK BTimeAway
  Fields
    AQSN : TAQSN
    sXML /"Xml string for easy alien router programming":string[2000]
    sDisplay /"display for summery screen" : string[1000] {part of xXML}
    sYear /"Start Year 1901..9997,DK,EMPTY
    eYear /"End Year " : 1901..9997,DK,EMPTY
    sMonthSeason /"Start Month/Season" : 1..24,DK,EMPTY
    eMonthSeason /"End Month/" : 1..24,DK,EMPTY
    sDay / "Start Day":1..43,DK,EMPTY
    eDay / "End Day ":1..43,DK,EMPTY
    Reason / "Reason for Time Away": 1..5
    DKWhen : 0..1
    DKWhenMonths : 0..12
    DKWhenWeeks : 0..3
    DKWhenDays : 0..31
    Notes / "Event Notes": string[200],DK,EMPTY
ENDBLOCK

TimeAwaySection1 : Array[1..5] of BTimeAway
TimeAwaySection2 : Array[1..5] of BTimeAway
TimeAwaySection3 : Array[1..5] of BTimeAway
TimeAwaySection4 : Array[1..5] of BTimeAway
TimeAwaySection5 : Array[1..5] of BTimeAway
```

The second step is to create a C# form to work with the Blaise data structure.

- Data grid view controls are utilized to represent domain cells
- Panels with different controls are used for data entry
- Other month and date user controls are utilized for easy month/day selection
- .net dataset is deployed for easy data storing and retrieving ; the xXML field is read into the dataset

Figure 14: C# Form Design View

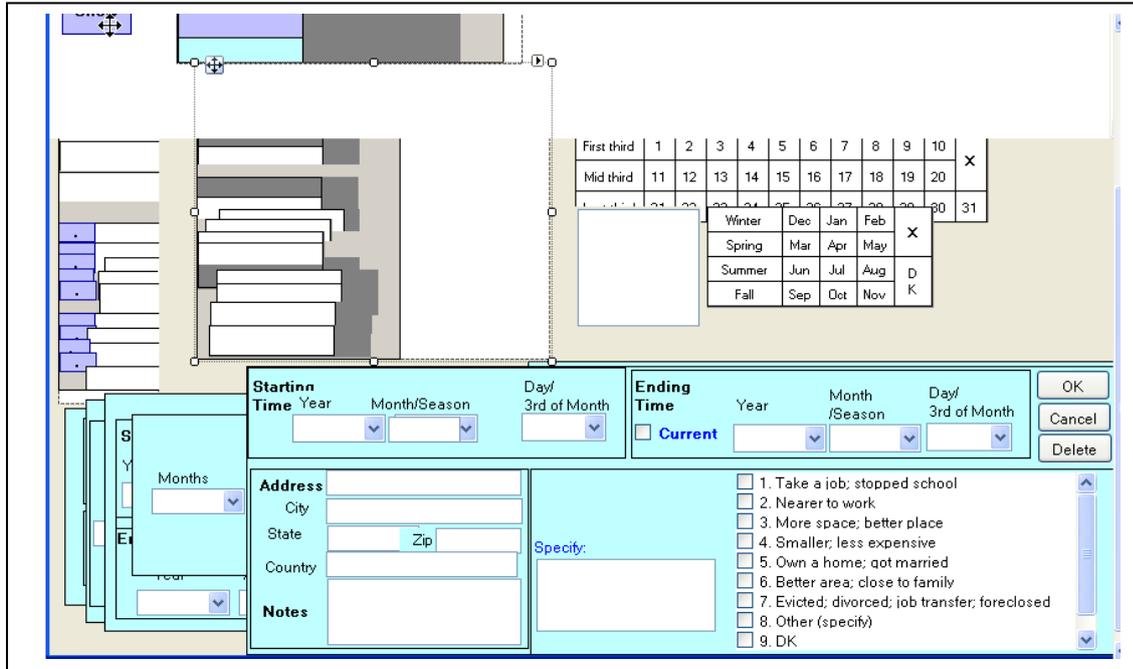
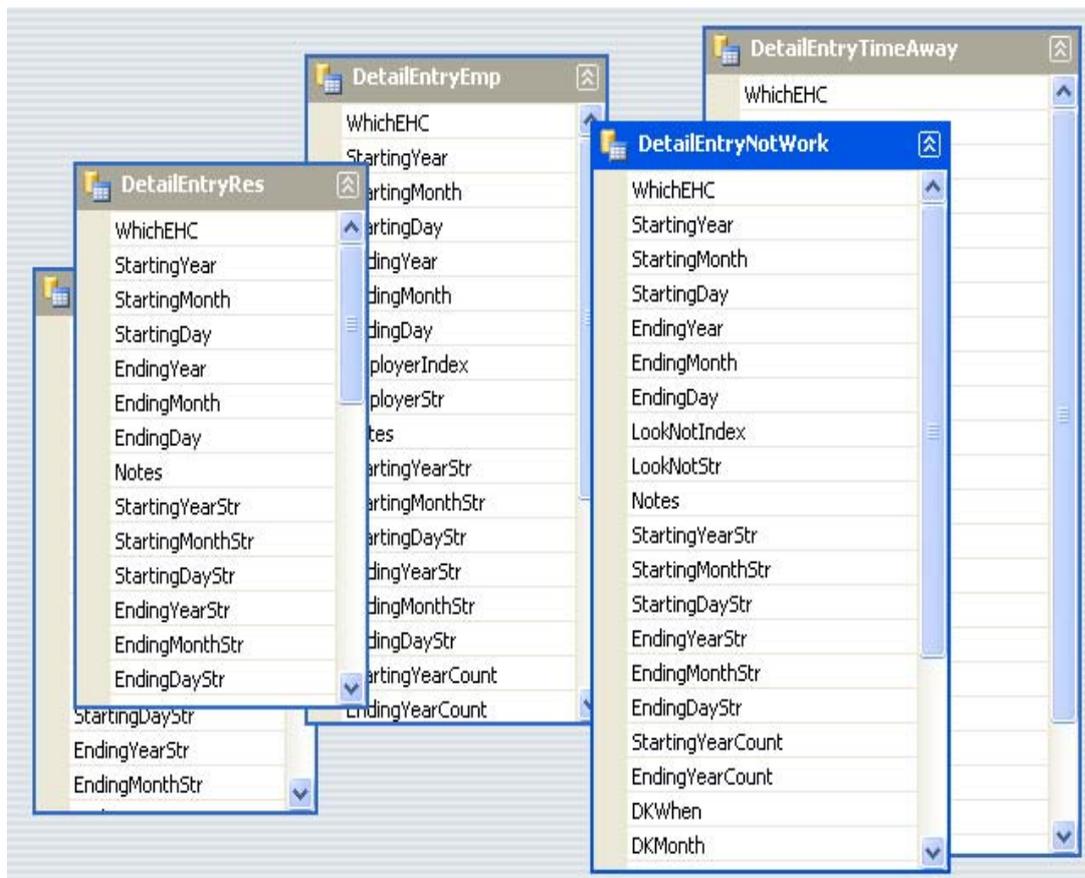


Figure 15: Dataset for all Domains



The final and critical step is to save and check the EHC data for consistency. Once the interviewer presses 'Cross Check Timelines', the program will save data to Blaise bdb. At the same time, it performs various consistency checks across time lines in all domains.

For example, NotWorkValid is used to check for overlapping periods between NotWork and Employment lines. The .net program will set NotWorkValid to '1' if it fails the check and then passes the value to Blaise. In Blaise, the following consistency check (Figure 16) is triggered upon exiting EHC.

Figure 16: Check for overlapping Timelines

CHECK NotWorkNotValid = 0
InVOLVING (BCDE1, EmployEHCRouter.EHCINTRO)
 "One or more Employment and Not Working lines overlap for more than one cell, please reconcile and correct."

Upon exiting EHC, the DLL sets the entry field value in Blaise to '2', so it prevents EHC from opening automatically when interviewer lands to the field next time. If interview wants to edit the EHC, he/she will need to change the value to '1'.

In addition to saving data to Blaise blocks, the data is also saved to an xml file. The xml is essential because the Blaise audit trail function does not capture any data collected in an external application.

8. Conclusion and Summary

Event History Calendar data collection may be superior method over the standard question list method for certain types of question series. The challenge for the PSID project has been learning by trial and error how to

take advantage of this collection method without compromising data quality and integrity. Based on our past experience developing an early version of an EHC for employment, we made a number of improvements including visual clues, consistency checking and clear warnings to the interviewer about potential problems. We have also incorporated 'on-the-fly' verification checks with the respondent so that we have a complete and clearer picture of their employment history.

Data conflicts can be resolved with the help of the respondent during the data collection process. This will mean more accurate coding of weeks of employment and out of the labor force spells and will reduce the numbers of hours spent post-processing trying to resolve common consistency errors.

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>

Beaule, A. Leissou E. and Liu. (2007): Experience Using and Event History Calendar in the Panel Study of Income Dynamics, Proceedings of the 11th International Blaise Users Conference, Maryland, United States, September 2007.

Belli, Robert F. and Stafford, Frank P., Calendar and Time Diary Methods: Measuring Well-Being in Life Course Research, Sage Publications 2008-09.

Belli, Robert F., Stafford, Frank P. and Shay, William P. "Event History Calendar and Question List Survey Interviewing Methods: A Direct Comparison." Public Opinion Quarterly, Vol. 65, p. 45-74, 2001.

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.

Time Use Diary Data Capture System built in Blaise and Maniplus

Fred Wensing, Softscape Solutions, Australia

1. Introduction

Time use surveys generally make use of paper diaries that respondents fill in over a specified period. The activity descriptions in these diaries need to be captured in electronic form and coded so that data can be analysed.

This paper describes a time use diary data capture system built for Statistics New Zealand using Blaise and Maniplus.

2. The Time Use Survey

For every household selected in the Time Use Survey, up to two adults are selected to participate in the survey. These persons are required to complete a hand-written diary of all the activities they do within a 48-hour period. Different days are allocated to different persons to ensure that all days of the week are covered.

As well as a primary activity, participants may record up to three additional activities which they may be performing at the same time. The diary also has provision to record where the activity is taking place, for whom and with whom the activity is done.

When the completed diary is collected, a supplementary Blaise questionnaire is administered by an interviewer to collect other socio-demographic information which will later be used in the analysis.

The paper diaries are returned to the office for data capture.

3. System description

The main components of the Time Use Data Capture System are:

- A Blaise instrument that is used to record diary entries and assigned codes
- A central register that keeps track of the diaries and progress with coding
- A coding list of activities and synonym entries
- A high level interface to provide entry to the system
- A user interface that assists with the data capture and double entry functions
- An interface to assist with the marking of dual coded entries

For quality control purposes a selection of diaries is coded a second time by another coder. To support this dual coding function a parallel environment is set up using a duplicate set of programs. Dual coded diaries are compared and discrepancies checked. Provision exists to produce a corrected record from the marking process.

4. The Blaise data capture instrument

The Blaise diary data capture instrument starts with header information such as the identification of the respondent, the allocated days and dates for the diary. Basic demographic information obtained from the supplementary questionnaire is also recorded in the diary instrument.

Once the header information has been entered the episodes can be entered one at a time (in sequence). Each episode consists of one or more activities which are entered using literal text strings and a trigram lookup on the activity code list to locate a synonym of the literal string. Once a synonym has been selected then the corresponding code is inserted by the system into the episode. Other details such as the location of the activity, for whom and with whom the activity is done are also recorded for each episode.

The instrument makes provision for 24 or 48 hours to be captured based on the designated days and actual recorded diary days.

Operators have a choice of entering the data in one of two methods:

- The Form approach - one in which each episode requires a full page of entries (or form) to be completed.
- The Table approach - one in which each episode is recorded in a row in a table.

For the Form approach (see Figure 1) the vertical split screen is used so that the whole episode can be readily seen. The identity and basic characteristics of the respondent are also visible in the Infopane where the corresponding activated question (and categories) is visible.

Figure1. Form approach to capture episode data

The screenshot displays the 'Time Use Survey Diary' application window. The interface is split into two main vertical panes. The left pane, titled 'Infopane', contains respondent information: Household: R289687003, Person: 1, #Members in HH: 1, Interviewer ID: 123, Respondent Name: Michael Palin, Age: 58, Sex: Male, Paid work: No, Occupation: (blank). Below this is a question: 'Activities for Episode[2] Where were you OR how were you travelling?' with a list of radio button options from 1 to 99, including 'At home', 'Travelling by car, motorcycle, truck or van', etc. The right pane contains a form for a single episode. It includes fields for 'Start' (Day: 1, Time: 8:00 AM), 'Stop time' (8:30AM), and 'Stop day' (1). A section titled 'Activities' lists four entries: 1. Literal 'eat breakfast', 2. Literal 'Newspaper', 3. Literal (blank), and 4. Literal (blank). Each entry has sub-fields for 'Activity' and 'Org(s)'. At the bottom of the right pane are fields for 'Location', 'With?', 'Codes' (11411, 44311), and 'Next'. The status bar at the bottom shows '3/125' and 'DiaryData.e[2].Where Dirty'.

Short-cut keys are provided for common entries like sleep and work and other functions needed for coding such as clearing an entry or providing a new literal string.

The Table approach (see Figure 2) enables the operator to see more than one episode at a time and for table functions to be used to insert, split, merge or delete entries, which would otherwise be difficult to do. Short-cut keys are provided to activate these table functions.

The preferred method of data entry is to use the Form approach. No matter which method is used, however, the same information is recorded for each episode. Entries under the two methods are synchronised and changes made under one approach are immediately transferred to the other. The active method is noted on the parallel block tabs.

Summary data is produced within the instrument for each 24 hour period to assist with edits and to identify missing entries (eg. of eating or sleeping).

Edits are provided within the instrument to ensure that episodes are entered properly and that full days of entries are recorded. Problems identified by the edits are located using the **Navigate show all errors** menu item. The status of each edit check is also 'piped' into a series of flags to enable the system to produce a text report of the edit failures.

Figure 2. Table approach to capture episode data

The screenshot shows the 'Time Use Survey Diary' application window. At the top, there are menu options: Forms, Answer, Navigate, Options, Help. Below that is a toolbar with buttons: Method, Insert, Delete, Split, Merge, Clear (F6), Literal (F7), Sleep, Work, Avail for care, Education. The main window title is 'TUSDiary | Diary Data (Passive) | EQFields | Table Data (Active) | SummaryDay1 | SummaryDay2'. Below the title bar, there is a summary section: Household: R289687003 Person: 1 #Members in HH: 1, Interviewer ID 123, Respondent Name: Michael Palin, Age: 58, Sex: Male, Paid work: No, Occupation: (blank). Below this is a text prompt: 'Activities for Episode[1] On what day did the episode start?'. The main area contains a table with columns: Day, Time, Stop time, Stop day, 1. Literal, Activity, Code 1, Who for, Org(s), 2. Literal. The table has 11 rows, with the first two rows containing data: Row 1: [1] 1 4:00AM 8:00AM 1 Sleep Sleep 11211; Row 2: [2] 1 8:00AM 8:30AM 1 eat breakfast eat breakfast 11411 Newspaper. The status bar at the bottom shows '1/11 DiaryTable[1].Start_Day: Dirty'.

5. Central register

In order to manage the data capture processes, the system has a central register which consists of a Blaise database and a management interface (see Figure 3).

The register is loaded with the details of households involved in the survey and whether there are diaries to be captured and coded. The register instrument starts with the Household identifier and a few summary attributes of the household such as its size. It then holds the identities of various persons involved with the household such as the interviewer, allocated coder, actual coder, dual coder and marker.

Figure 3. Register and management interface

The screenshot shows the 'Data Processing Manager Task List' interface. It features a table with columns: Household, Status, Status Date, Alloc. coder, Actual coder, O, Dual coder, Marker, Status Diary 1, Status Diary 2. The table lists 12 households with their respective statuses and dates. To the right of the table is an 'Allocation' section with buttons for Person, Coder, Dual coder, and Marker. Below the table is a search bar and a 'Key type' dropdown set to 'Primary key'. At the bottom, there are 'Actions' (Details, QA select, Non-Resp, Re-select), 'Filter' (Define, Refresh), and 'Reports' (HH_Status, DP_Workload) buttons. A status bar at the very bottom shows: 'Total:12 Loaded:9 Assigned:1 Coding:2 Dualing:0 Marking:0 Done:0 Non- resp:0 Export:0' and an 'eXit' button.

The Register instrument also holds information about the one or two diaries that were completed by respondents from that household. The status values of each diary are updated directly by the programs which manage the diary coding, double entry and marking processes. The overall processing status of the household is then derived from the status values of the two diaries.

Most interactions with the Register data file are from the (Maniplus) programs which manage the various processes in the system. The main process which can be used to make selected changes to the status or allocation of persons is the Register management interface which has buttons that can be used to activate the various tasks.

6. Coding list of activities

The captured activities are coded to a structured 5-digit code list maintained as an external file to the diary instrument.

Once the literal string for an activity has been entered then the system looks-up the code list for that text string and assigns the corresponding code.

If the literal string is not found in the activity code list then the operator can do one of two things:

- Attempt to use a synonym description, which triggers a trigram search of the activity coding list to find a suitable result, or
- Lodge a new (uncoded) literal string into the record for subsequent decision by those who maintain the activity coding list

The Diary data capture system provides an Activity maintenance interface (see Figure 4) to enable various code maintenance functions to be carried out.

Figure 4. Activity code list maintenance interface



The Activity code maintenance functions include:

- Extracting a list of activity descriptions which have no code
- Importing a new list of activity descriptions and synonyms
- Identifying mismatches between old and new versions of the activity code list
- Running a program that will update previously uncoded activities by re-applying the look-up process against the new code list

7. Main interface to provide entry to the system

The system is entered through the main interface which gives access to the various processes that are to be carried out.

The main interface is supported by a list of authorised users which determines their role in the system (Data Processor, Quality Manager or Data Processing Manager). Depending on the role of an operator, the interface will provide access to more or less functionality (see Figures 8A and 8B).

Figure 8A. Main interface as seen by a data processor

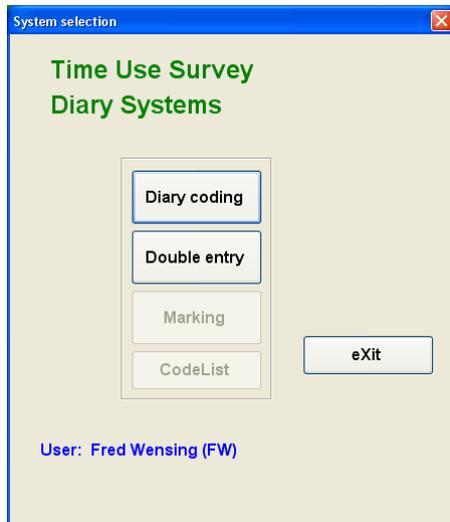
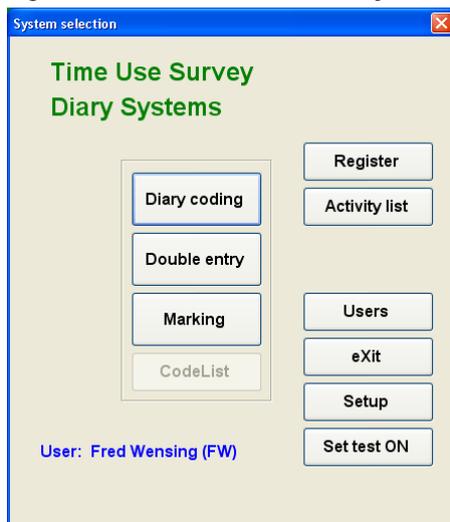


Figure 8B. Main interface as seen by a manager of the system



The list of users also contains a checking rate, for each user, which identifies the percentage of diaries that are to be double entered for quality assurance. These rates may be adjusted by the Data Processing Manager from time-to-time based on quality outcomes.

The main processes and sub-systems that are accessible through the main interface are:

- Installation of the local activity code list (whenever it needs to be updated)
- Diary coding
- Double entry
- Marking of dual coded entries
- User maintenance
- Activity code list maintenance
- Diary register maintenance
- Configuration of the system (set-up).

8. An interface to assist data capture functions

The data capture interface starts with a list of households that have been assigned to the operator for coding. To make it easier for an operator to locate a particular household in the system, the list can be filtered on the basis of diary status values and/or month of collection. Search on household number is also provided. It is also possible to enter a household which is not on the list, provided that it is in the system and not being used by another operator.

Once a household has been selected for data capture the operator is presented with a screen that contains the known information about the household (see Figure 9) and the status of diary data entry to-date. The interface also contains buttons that give access to other information and reports.

Figure 9. Data capture management interface

Details of selected household

Household ID: R289687006H001 (3 persons) View

Selected person 1
Name : Fred Jones
Male aged 28
View

Selected person 2
Name : Ted Jones
Male aged 18
View

Diary 1
Status: Partially coded (<48 hours)
BL status: Dirty
Episodes: 0
ED report
Edit

Diary 2
Status: Not started (but expected)
Create

eXit Report View

You will notice from Figure 9 that the details of up to two selected persons and the status of diaries are shown. This is consistent with the survey design which required up to two persons to participate. The household diaries are always to be captured as a set because there may be interactions between the respondents that can affect the resultant coding.

To assist the coding staff to understand the context of activities that respondents have reported, access is provided to the full supplementary Blaise questionnaire and Household questionnaire information through the View buttons. Data from those sources cannot be changed by the operator. A Report button is also provided to produce a summary of the household situation (see Figure 10 for sample report) which can be printed if necessary.

9. An interface to assist with marking

For diaries that have been double coded the system compares the matching diary entries and identifies all the discrepancies. These records are then available to the quality manager to carry out a marking process where he/she checks each discrepancy and ticks off the correct or best entries.

The interface for marking is similar to that for data entry but there is more functionality (see Figure 13).

Figure 13. Marking management interface

Diary marking for this household

Household
ID: R100010002 (4 persons) View

Selected person 1
Name : Andrea Green
Male aged 74
Person 1 View

Selected person 2
Name : Bob White
Male aged 62
Person 2 View

Diary 1
Status: Partially coded (<48 hours)
Time diffs = 1
Code diffs = 1
Episode diffs = 1
Mixed: Edit Mark
Hybrid: Create Edit

Diary 2
Status: Not started (but expected)
Time diffs =
Code diffs =
Episode diffs =
Mixed: Edit Mark
Hybrid: Create Edit

eXit Report View

From this interface the operator can examine the matched episode records (using the Mark button). This brings up a screen which shows the discrepancies, one by one, and provides tick boxes to identify the correctly coded entry (see Figure 14).

Figure 14. Marking screen for one mismatched episode

Episode for scoring

Episode details
#2 8:00AM Day 1 8:10AM Day 1 8:30AM Day 1

Activity 1: Coder (A) Check washing Code: 3121 Who for: 10
Dual coder (B) Read personal letter Code: 4231 Who for: 10

Activity 2: Code: Who for:

Activity 3: Code: Who for:

Activity 4: Code: Who for:

Previous eXit Next

For the entry shown in Figure 14, there is a discrepancy in the end-time (shown in red) and a discrepancy in activity 1. The operator can tick the corresponding boxes on the right-hand side to identify whether coder A, coder B or Neither are correct.

Once the marking has been completed it may be possible for the operator to prepare a hybrid record that combines the correct responses from the matched episodes. This will enable the most correct data to be used, particularly while the coding is in its early days.

10. Technical challenges

There were a number of technical challenges that needed to be handled in such a complex system. Some of these are discussed here.

10.1 Local deployment

The most efficient way to operate Blaise data capture, when there is a complex instrument and a fair amount of external look-up steps, is to place the Blaise data capture file and the external look-up files local to the operator.

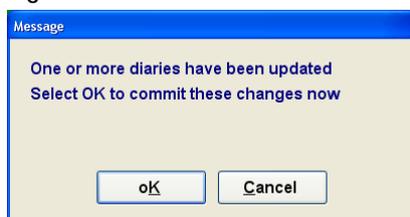
The two issues that needed to be handled as a result are:

- Ensuring that the activity code list is always up-to-date
- Transferring the diary data to and from the central data store

The consistency of the activity code list is managed by recording the version number of the current activity code list in the set-up settings for the system. The version number of the activity code list in the local environment is then checked whenever the system is started. If the version is not up-to-date then the main data entry access buttons are de-activated until the latest version is installed.

The Diary data capture system has a central data store for each process (diary coding, double entry and marking) which is placed on the network. Once a household has been selected then the system copies any existing data to the local disk drive before enabling the operator to change it. At the end of involvement with that household the system prompts the operator to commit the changes or go back and revert to the original version.

Figure 15. Exit screen when data has been changed



10.2 Filtered lists

The system makes use of a single central register of the households involved in the survey. That register is located on the network and is accessible to all managers and operators.

Once the survey gets underway, the list of households is expected to become large so a filter technique has been added to enable the system to target households that have been assigned to a particular user, have reached a particular status or from a particular month.

The filter works by applying a series of conditions to the list of entries in the main register. Those entries which match the filter conditions are then copied to a temporary file before displaying them on the screen.

Using a filter frees up the register for multiple shared access. The register data only gets updated with status information when the operator closes a diary coding, double entry or marking process.

Figure 16 shows the filter definition screen used for diary coding or double entry. A more comprehensive filter is available for register maintenance functions.

Figure 16. Filter definition screen for data capture

10.3 Tracking time over 48 hours

The Time Use diaries record episodes and activities over a 48 hour period commencing at 4AM on the first day and finishing at 4AM on the third day.

To make it easier to manage these time events, and check the sequencing of episodes, the diary instrument has been set up to convert each time point (consisting of a clock time and day number) into an integer with values from 0 to 2880. These codes then represent the 2880 minutes in a 48 hour period. Each start and end time is duly converted.

Using this simple conversion makes it much easier to check times against each other (to ensure episodes are consecutive) and to calculate time differences or sums.

10.4 Providing for alternative data entry methods

As mentioned in section 4, the diary data capture instrument provides support for two methods of data entry, the Form approach or the Table approach.

In order to synchronise the two methods the instrument uses two arrays, one for each method, containing all the episode fields. The first array is used in the ordinary flow of the questionnaire and the second is used in a table. Both arrays are initiated using the KEEP instruction then a single field called EntryMethod (with two possible values) is used to control which one is the active method and which becomes the passive method.

Depending on which method is active, the Rules in the diary instrument copies all the values of the corresponding active array elements to the other (passive) array.

It should be noted that duplicating the episode data in this way does add some load to the execution of the instrument. On the other hand, having the table method available makes it easier to handle episode level actions of insert, delete, split and merge.

10.5 Table operations

The special block commands of INSERT and DELETE, along with other assignments, are used to insert or delete rows in the episode table and move the remaining data around to achieve the desired outcome (being either insert, delete, split or merge).

These special operations are controlled by setting a Flag to allow the corresponding rules to be executed once. The Flag is reset to the inactive value at the end of that section of the rules to stop the actions from being repeated.

These special table events are triggered using corresponding buttons placed on the Menu bar. Once pressed the program behind the button activates the Flag value and the action type and runs the Rules.

10.6 Checking the dual coding of activity episodes

When comparing the dual coding of activity episodes it is not a simple matter of comparing the elements of two arrays. That is because one array or the other may contain more or less entries. This can happen where one operator considers that two or more episodes have taken place for a particular time period, and another operator considers that only one episode has taken place for the same time period. There are many other possibilities for time discrepancies to arise.

So, before the detail of two sets of activity episodes can be compared, they need to be processed and re-aligned based on start times and end times.

A procedure was developed in Maniplus that compares the start times and end times of the two arrays of episode information and makes corresponding re-alignments to all the entries when problems are detected. The re-alignment process involves inserting dummy episodes that have the same start and end times (effectively an episode with zero elapsed time) and contain no activities. The episodes effectively push the subsequent mismatched episode up by one. The whole process is then repeated across all the elements of the array until all entries match on start time or end time or both.

11. Conclusion

Development of this complex Diary Data capture system has been a challenge but Blaise and Maniplus has proven to be versatile and flexible. The posters will reveal more of the system design and illustrate some of the source code.

Paradata and Blaise:

A Review of Recent Applications and Research

Jim O'Reilly, Westat, USA

Survey paradata, information about the process of survey data collection, has steadily increased in the breadth and depth of use. From modest beginnings in the 1980s as trace files of the fields entered, keys pressed, and timings during computer-assisted interviews, the value of the information for addressing significant challenges in the survey process has been explored and applied in increasingly diverse ways.

A number of survey organizations have recently implemented or enhanced major management systems with paradata as a central feature. As well, computer-assisted recorded interviewing (CARI) seems on the verge of becoming a powerful new form of paradata with the potential to qualitatively enhance important elements of the survey design and survey collection processes. In this paper we review these recent developments of paradata research and application.

At the 2009 FedCasic conference held in Washington DC a number of presentations focused on using paradata to monitor and control survey quality. Compared to similar sessions in prior years, the presentations this year seemed notable in terms of the maturity of the processes and the production experiences described.

PANDA

Ari Teichman (2009) of the U.S. Census Bureau discussed his agency's Performance and Data Analysis (PANDA) system and its approach to improving survey quality. This comprehensive management system has been implemented for the 2007 American Housing Survey. Key goals of the system are to provide early warnings of interviewer difficulty with key survey concepts and possible falsification.

The PANDA system provides paradata-based reports to managers at every significant level of the project on key production metrics – interview duration, interviews by time of day, interview result, outliers, and more. Among the indicators of possible falsification are high rates of vacant housing units, small household size, interviews conducted at unusual times, short interviews and others. Management reports include summary level reports on Regional Office/Area Totals, Cumulative/Weekly Report, and Average Cases Per Interviewer; and interviewer reports on Highest Non-response Rate for Salary of Reference Person; Highest Number of Regular Occupied Interviews Completed in Less than 20 Minutes; Highest Number of Cases Completed 12:00 a.m. – 7:59 a.m.; Highest Vacancy rates, and many more.

The system also provides the field managers with the ability to drill down in trace files to examine details of interviews. Field managers reported that the most useful features for them were identifying cases completed between 12am and 8am, targeting potential falsification, and searches and access to trace files. Field managers also said they used the system to search for detailed information of interviewers' work, to address potential problems appropriately, and to identify interviewers needing retraining or cases requiring re-interviews.

Teichman said the system have been well accepted by staff at various organizational levels and is being implemented in other major Census surveys beginning with National Health Interview Survey.

Statistics Canada—Paradata as a Research Cornerstone

François LaFlamme (2009) of Statistics Canada discussed her organization's major commitment to data collection research using paradata as the cornerstore. The goals are to understand the process, develop new efficiencies, evaluate new initiatives, and maintain and improve data quality. Data collection is a top concern

because it significantly determines data quality and accounts for nearly half to three quarters of total survey expenditures.

Statistics Canada has developed a paradata warehouse to store call and contact information for both telephone and in-person interviews and administrative and payroll information. As a result, the major cost of the paradata system, the data warehouse, is centralized, minimizing the burden on individual studies and customers, and insuring that all surveys are represented. This, in turn, enhances the research potential and benefits. As well, the paradata system provides in-depth and timely cost information for survey cost analysis.

LaFlamme discussed some early research efforts focusing on CATI and using paradata from a variety of surveys--RDD, cross-sectional, longitudinal, social and agricultural. Analyses were done on the time for contact attempts and system work, contact rates and calling patterns, and the production-cost relationship.

Strategically, Statistics Canada foresees important opportunities for improvements using the system to research such issues as:

- Better use of pre-collection information and information gathered during collection
- Methods used after the first contact
- Development of a responsive design framework combining active management and adaptive data collection
- Predicting collection requirements during collection based on progress metrics

LaFlamme reported on studies of production-cost relationships to provide more useful survey productivity indicators. One example is an analysis of the impact of capping the number of calls on survey cost, which found maximum potential cost savings from 3.1% to 4.2% for longitudinal surveys.

LaFlamme said research is underway on using responsive design for CATI surveys -- adapting the collection process dynamically based on on-going information on the productivity and costs of the surveys. Also, the calling process would be modeled and managed to maximize the likelihood of making contact and conducting an interview. Staffing would also be planned and predicted based on survey progress.

Statistics Canada -- POINT

Mike Maydan (2009) of Statistics Canada presented a thoughtful discussion of the management challenges of the increasingly complex survey environment. In the Statistics Canada regional collection structure, there are multiple data collection, archive, and reporting systems with competing priorities, needs, and dimensions.

To improve coordination and integration, standardized survey reports has been developed based on by case-level paradata for CATI (from the Blaise telephone history file) and CAPI case management events--timing, routing, and outcome details. This provides managers and supervisors with measures of data accuracy with response/non-response rates, non-response follow-up, refusal conversion, and tracing.

Intra-case paradata—audit trails—enables evaluating the “act of collection”, enhancing the evaluation of interviewers. A relatively new application of this paradata is the POINT (pace of interview) system, which focuses on identifying irregular production calls. The system is designed to apply objective performance measures to help identify interviewers for possible retraining or other action. The two indicators are the pace of the interview (field changes per minute) and item non-response (don't-know and refusals).

The level for potentially significant irregular call levels is based on the survey's early collection experience--six hundred calls with at least twenty content field changes. The threshold is 175% of the early collection mean for field changes per minutes and greater than 25% item-non-response.

Reports based on paradata through the previous day provide statistics on the number of interviewers, calls, field changes, interviewing pace, item non-response, and irregular calls by site and interviewers. Audit trails are also used by managers for repairing technical data loss, for post-collection validation and verification, and for time per section report for post-survey analysis and budgeting.

National Center for Health Statistics – Public Use Paradata and Research

Beth L. Taylor (2009) of the National Center for Health Statistics (U.S.) discussed the public use release of paradata on the data collection process along with the standard public use data file of the 2006 National Health Interview Survey. This appears to be the first time paradata has been included in a public use data release.

The National Health Interview Survey (NHIS) is an annual survey of 35,000 families conducted in-person with telephone follow-up. Detailed contact history information are collected on each visit attempt: description of the attempt, whether reluctance was encountered, and strategies used to complete the interview. For non-contacts the contact description and strategies are kept.

Also, the paradata includes the interview's language, cooperativeness of respondent, interview mode, reasons for interview breakoffs, type of non-interview cases, time of interview and module/section times. From the audit trail comes the time per question, dates, and interviewer notes.

To protect confidentiality and improve usability of the data, various items have been recoded to preclude individual identification. The paradata can be used alone or linked to health data. Examples of possible analyses include:

- Contact attempts and interview completion
- Time of day of interview
- Interview strategies and successful completion
- Characteristics of hard-to-contact families
- Modeling impact of interview mode on health outcome

With the release of the NHIS 2008 public use files, additional items will be added on visit attempts and use of function keys and language of interview. Also, working with the Census Bureau, which administers the NHIS, the PANDA system has been enhanced in the areas of interviewer performance, tracking, and reporting.

Public use of the NHIS paradata so far seems limited to survey methodology students. However, James Dalheimer of the NHIS group said in a recent conversation that internally there is extensive study of paradata around the issue of interviewer performance, using both case level contact histories and audit trail item and section times.

One current emphasis is the process of selecting cases for reinterview. This has been usually been done randomly with very few targeted supplemental reinterviews based on suspected problems. Research is underway on applying statistical predictors for re-interviews. Other research of interest based on paradata is:

- non-response adjustment to weights
- sub-unit response
- high-effort interviews and bias

Summary

These presentations indicate how far the use of paradata has advanced. Not long ago, the main analytic efforts were on question and section timings, looking for outliers and anomalies that suggest problematic interviewer or instrument performance. Now, organizations are integrating paradata into their core management process, providing carefully developed metrics and reports to various supervisory levels, as well as giving direct access to detailed audit trail information for line supervisors.

Another indicator of the expanding importance of paradata is that for the first time there will be an invited-papers session, "The Use of Paradata in Federal Government Surveys", at the annual Joint Statistical Meetings in August 2009. The papers are:

- "Use of Paradata to Manage a Field Data Collection", Robert Groves (University of Michigan), et al.,
- "Using the Fraction of Missing Information to Monitor the Quality of Survey Data", James Wagner (University of Michigan)
- "Modeling the Difference in Interview Characteristics for Different Respondents", John Dixon (Bureau of Labor Statistics)

- “An Evaluation of Nonresponse Bias Using Paradata from a Health Survey”, Aaron Maitland (National Center for Health Statistics) et al.
- “Subunit Nonresponse in the National Health Interview Survey (NHIS): An Exploration Using Paradata”, James M. Dahlhamer National Center for Health Statistics and Catherine M. Simile (NCHS)

CARI -- Paradata’s Next Generation

Computer-assisted Recorded Interviewing (CARI) provides a direct audio recording of the interviewer and respondent during the interview, enabling the review and analysis of a multitude of potential facets of the interaction, far beyond what audit trails and trace files provide through time stamps, data field changes and other keyboard actions. As with any significant new method CARI’s development and application has emerged and grown over a number of years. The most frequently used CARI capability has been for quality assurance (QA) focused on verification that the interview took place and evaluation of the quality of the interviewer performance.

Among significant recent CARI research is the U.S. Census Bureau’s rigorous CARI field test evaluation, (Arceneaux, 2007). The study is part of broader research effort toward implementing “CARI into all of the Census Bureau’s computer-assisted personal interview (CAPI) surveys.” The 2006 study was conducted with a sample of cases selected from three regions. CARI interviews were recorded for 423 cases. The test found that CARI did not impact production data, the CARI technology functioned properly, recording occurred without detection, and technical problems did not increase. Respondents were very receptive to CARI while interviewers were mixed with 39% comfortable and 23% opposed.

Two negative findings were:

- Recordings were judged to be high quality (rated excellent or good) for 85.6% of cases, while the desired level was 96%.
- The HWS response rate was 81% compared to 90% on a comparable sample from the National Health Interview Survey.

Arceneaux mentions a number factors in the field test study that may mitigate these differences and recommends further research to advance adoption of CARI for “all of the Census Bureau’s CAPI surveys.”

In the 2008 American National Election Survey (Lupia, et al., 2009), CARI was used to verify that pre-election and post-election interviews were completed. CARI files were transmitted overnight to Research Triangle International (RTI).

“Cases were reviewed using a process that focused on address validation, respondent self-identification, confirmation of both respondent and interviewer voices, and consistency of voice(s) across the recordings. Any concerns with one or more of these dimensions prompted further review, including in-person validation in the field if needed.”

The percentage of cases reviewed was adjusted during different phases of data collection: 100% during the initial interviewer incentive period and then at least 10% of completed cases per interviewer. Indications of possible quality or validity problems prompted review of all completed cases for the interviewer. CARI refusal rates were also monitored closely.

Wendy Hicks (2009) and colleagues at Westat and the National Center for Health Statistics presented at the FedCasic Conference their application of CARI in the evaluation of questions and interview performance in a national health survey. They described a full system approach to CARI designed to meet varied research objectives.

In the central office system CARI recordings are linked to the survey data and audit trails. The system permits flexible review in different modes, in which coders can review one question across a series of interviews, or a series of questions in an individual interview. The system provides coding with a structured tool, links to the survey data and operational process data, other built-in quality control functions, and a variety of reports.

Hicks presented statistics on some important survey questions, examining whether a given question was read with a major change, whether the respondent asked for clarification, and whether the response did not match the format expected. Other results presented for selected questions included whether the respondent asked a question

or the answer did not match the format, comparing household and institutional types of respondents. Another report focused on interviewer skills and the percentage of interviewers with difficulty in terms of minor changes from verbatim, major changes from verbatim, not probing when needed, and using leading probes.

The Hicks presentation was greeted very positively by many at the conference. The combination of a comprehensive and effective system to integrate the management, processing, and coding, along with valuable analytic findings, appeared to interest many participants in the potential of CARI.

Another intriguing indicator of the rising interest in CARI as a major new tool for surveys came at the 2009 IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology. Officials from the Office of Behavioral and Social Sciences Research of the National Institutes of Health in discussing the NIH BIG Data Initiative for surveys of the future cited CARI as one of four notable developments (Mabry and Philogene, 2009).

Finally, the interest and application of CARI is likely to be enhanced by the about-to-be-released implementation of CARI in the Blaise system. As a result the primary survey interviewing system for complex surveys will have CARI built-in and will support highly flexible and dynamic CARI methods and functions.

Final Thoughts

The evolution of paradata in survey research seems to be reaching an important threshold. Technology, organizational systems to integrate the key processes, and research demonstrating the value of paradata to better understand important and challenging survey elements seem to be aligning. The stage seems to be set for much wider adoption and utilization not only by the largest organizations, but also by other groups with similar needs.

References

- Arceneaux, Taniecea A.. 2007. "Evaluating the Computer Audio-Recorded Interviewing (CARI): Household Wellness Study (HWS) Field Test". Proceedings of the Survey Research Methods Section, American Statistical Association
- Hicks, Wendy., Brad Edwards, Karen Tourangeau, Laura Branden, Drew Kistler, Brett McBride, Lauren Harris-Kojetin and Abigail Moss. 2009. "A System Approach for Using CARI in Pretesting, Evaluation and Training" *FedCasic Conference, Washington DC*
- LaFlamme, François. (2009) "Overview of CATI Data Collection Research Focused on Developing Operational Strategies for Process Improvement". *FedCasic Conference, Washington DC*
- Lupia, Arthur, Jon A. Krosnick, Pat Luevano, Matthew DeBell, and Darrell Donakowski. 2009. *User's Guide to the Advance Release of the ANES 2008 Time Series Study*. Ann Arbor, MI and Palo Alto, CA: the University of Michigan and Stanford University.
- Mabry, Patricia L. and G. Stephane Philogene, 2009. "Systems Science Methodologies To Protect and Improve Public Health". *IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology, Nashville TN*.
- Maydan, Michael J. (2009) "Using Paradata to Monitor Survey Quality in Statistics Canada's Regional Data Collection MIS Reports". *FedCasic Conference, Washington DC*.
- Taylor, Beth.L. 2009 "The 2006 National Health Interview Survey (NHIS) Paradata File: Overview and Application". *FedCasic Conference, Washington DC*.
- Teichman, Ari. 2009. "Panda: Using Paradata to Improve Data Quality". *FedCasic Conference, Washington DC*

A Systematic Approach to Debugging in the Blaise Environment: An Author's Perspective

Peter Sparks, The University of Michigan

1. Introduction

There has been much written about testing Blaise instruments using a variety of systematic approaches, databases and logs of changes, version control, automated instrument creation, testing cycles, analysis utilities, and so forth. But what about the author writing programming code that follows the survey specifications? What tools and methods are available at the author's fingertips to make this development quick, robust, and accurate?

Blaise provides a rich environment for development of survey instruments, Manipula/Maniplus scripts, Cameleon, Basil, CATI, and CAWI. The author has access to the source code, metadata, test data, as well as the modelib, configuration files, datamodel properties, "watch window," menu files, keystroke files, Manipula debugger, and Blaise API (Blaise Component Pack), and so on. Additionally, DLLs (dynamic link libraries) can be developed to enhance the existing tools. For all of this, though, debugging can still present a challenge.

This paper presents methods for writing Blaise code (instruments and scripts) that will make later debugging easier, as well as a practical approach to using the existing tools to aid in debugging, testing, and the creation of new tools using the Blaise 4.8 environment.

2. Standards to Minimize Debugging

"A good offense is a good defense." The first steps to debugging should always be to reduce the amount of checking needed in the first place. The points below have been thoroughly covered in other materials and are just provided as a reference.

2.1 Clean and Standardized Specs

Ideally, every survey should have clear, standardized specifications that clearly define everything that is to be programmed. This is used by everyone involved and is the final authority for changes. In general, the specs should cover

1. Defined logic flow for all values including missing
2. Ranges defined
3. Checks/signals defined
4. Checkpoints defined
5. Fills defined
6. Question text layout
7. Help, booklet, and other references

2.2 Standardized Programming Conventions

The programmer, likewise, should follow standard coding conventions. Not only does this make it easier to isolate problems, but fellow programmers can more readily assist in case of a difficult problem without overcoming the additional burden of poor coding. This, of course, is not a complete list.

1. Source code formatting
2. File naming and content
3. Headers and comments
4. Variable naming
5. Block and Procedure use

6. Layouts

3. Write Easier-to-Debug Blaise Code

The steps needed to write better code are not hard, but do require discipline, and in fact heavily rely upon consistency. The points given in 2.2 are not trivial, but they do have tremendous results when done well.

In general, first follow the coding standards for your organization. Then apply these concepts as you are able within those confines. The items below are focused on Blaise as the programming language.

1. Create each section as a separate block and file
Physically separating each top level section as its own file makes it easier to work on, track in version control software, and is a logical unit by itself. Do not put more than one section in the same file.
2. Put a comment header on each file. Include revision dates.
This allows a revision history to be maintained, and if the survey is printed also gives more information.
3. Don't mix Fields, Auxfields, Locals multiple times
The key is that you want to be able to locate information quickly, and having a jumble is self-defeating, even when working with the Control Center Explorer.
4. "Forward" passing of assignments
If you don't use parameters to blocks, then do assignments to blocks "below" the current level instead of grabbing an assignment from down below. The "above" will be consistently on the rules path and will not create a "generated parameter."
5. Minimize overuse of procedures
Procedures do cause the survey to run slower, and procedures embedded within procedures can quickly multiply and cause problems. Because procedures do not store data they also can be harder to debug, and documenting procedures is also more difficult. Use your best judgment.
6. Use fields for potential data items (counts, internal checkpoints)
Although the client may not think of it, always consider using a field for counts of things (family members, children, item lists, ...) instead of auxfields/locals. The data will be stored and available instead of having to be recreated (potentially with errors) afterwards. Having the results stored in the data also make it easier to debug some routines.
7. Use auxfields for fills, some counts
You have more control over auxfields than locals, and as a bonus can see fills in the DEP watch window. Those counts that are easily recreated can be made as an auxfield instead of a fill.
8. Use locals for loop indices, or those items that always reevaluate\
Locals are always reevaluated if on route, cannot be seen by the watch window, and are never stored. Relegate them to loop indexes and sometimes counts.
9. Define parameters on blocks as needed
Parameters minimize the number of generated transit parameters. These have the highest overhead and are on-par with locals. If you do use parameters, be sure to explicitly define INPUT, EXPORT, or TRANSIT.
10. Always .KEEP fields that are assigned
Don't assume that a field will be kept correctly ... make it explicit at the point just before the assignment at the same level or one higher than the assignment. Implied KEEPs are at the bottom of the rules section, and sometimes not at the place you would expect.
11. Create fills and other assignments as local as possible
Don't put all your fills at the uppermost level and do assignments at the bottom, but define your fills for that block if possible. This will make it easier to track down and observe in the watch window.
12. Create internal checkpoints where used
Likewise, keep internal checkpoints (assigned variables used for logic pathing) as local to where they're used as possible. Whenever possible make these fields and not auxfields/locals.
13. Define ranges for everything

Don't be a sloppy programmer! The specifications should give explicit widths and ranges for everything. Define widths for fills like `STRING[28]` instead of `STRING` (implied 255 characters). Use ranges like `0..12` instead of `INTEGER` (width of 18!). When it comes time to work with the data you won't have to deal with impossibly long fields, and the analysts will appreciate it as well.

14. Create a type file/library

Put all your types into one place, name them consistently, order them consistently, and place them into one file. Anytime you have to modify a type you'll know exactly where it is, and you will not duplicate types unnecessarily.

15. Use comments

Comments are handy for matching very long `ENDIFs` to their parent `IF/ELESIF` statement. They are good for describing complex algorithms or special study constraints. They are good at documenting changes to an instrument after data collection has started. In fact, use comments as need but be sure to make them accurate. Place comments appropriately per your organization.

16. Unique language fills

If you work on multi-lingual surveys you have to program fills per language. I would heavily recommend that a distinct fill is created, named appropriately, for each language instead of reusing a fill for the languages. For example, create `xHeShe` and `xElElla` for the fills (he/she) and (él/ella) respectively. Otherwise you will need to heavily use the `ACTIVELANGUAGE` command in the rules, later documentation of your survey becomes more complex, and debugging each fill requires a debugging run in each language instead of just once for the main logic. Yes, it is a little more overhead per memory but a much cleaner approach with less coding.

4. Know Your Work Environment

One of the greatest obstacles to debugging and doing it well is knowing the tools available to do the job. The Blaise system is very large, complex, and covers many different platforms. To cover all the debugging features of Blaise in detail would create a very large paper. What follows is a discussion of the different environments, how it can be accomplished, and tips and suggestions for making better use of the feature.

4.1 Delta

This tool is an enhanced datamodel browser that can compare two datamodels. It provides a graphical rules viewer of the datamodel and allows the user to navigate through the datamodel according to the displayed `IF/THEN/ELSEIF` blocks in the rules. A selection of a node in the tree view also focuses the appropriate place in the rules viewer.

Delta's value is that it has a rich view of the datamodel rules, and that it can compare two datamodels, such as from one release and then a second. This is very valuable when source code is not available, or documentation of changes has not been up-to-date in the source code.

Delta is XML-based, and so new stylesheets can be used to view the information from the datamodel, such as the provided `ixml.xsl` stylesheet found under the `C:\Program Files\StatNeth\Blaise 4.8 Enterprise\Bin` directory. This means that the entire XML from Delta could be retrieved, or other stylesheets to be used by Delta can be created.

4.2 Blaise XSD schemas

This utility exports a schema for the entire datamodel that describes types, fields, codes, question text (and language identifier), ranges, data sequence of questions, and so forth related to the data of the datamodel.

Note: it is missing some details, such as the width of fields, checks/signals or other features found only in the rules.

This information could be a great source of information about the datamodel without resorting to writing special BCP utilities. The XML can be imported into databases, spreadsheets, or stylesheets written to take advantage of the information, and so forth.

See the Blaise Data Centre for the related data XML.

4.3 X-Tool

The Blaise help has this entry:

"X-tool is a software package for the analysis of experiments embedded in ongoing sample surveys. With X-tool you can test hypotheses about differences between design-based estimates of finite population parameters observed under different survey approaches."

Practically speaking, I don't have a current use for X-Tool in the context of debugging Blaise applications.

4.4 Manipula/Maniplus

Manipula has been the robust tool for extracting data and manipulating from a Blaise database, and most recently it works with datamodel information, and can be called from DEP and Basil.

It is incredibly versatile, and as a result the focus is on features that make debugging easier rather than features of the language.

4.4.1 Manipula Debugger

The debugger is basic. The greatest value is to step line by line through a script and examine variable via the "evaluate" dialog, tool tips, or the Manipula watch window.

The things you can do with the debugger are:

- Set/clear a breakpoint/clear all breakpoints
- Step over a procedure
- Trace into a procedure
- Evaluate a variable via a dialog & tool tip
- Display multiple variables (watch window)

4.4.1.1 Manipula watch window

This is not the same as the DEP watch window, and is set by a literal field name, typed, or inserted from the evaluate dialog.

4.4.1.2 Evaluate window

The fieldname has to be typed and a pick list of prior viewed fields can be selected. Valid fieldnames can be added to the Manipula watch window.

4.4.1.3 Cursor tool tip window

Hover the cursor over a variable while the script is in "stopped" mode and a small tool tip will display the variable's contents.

Remember that this debugger is only for Manipula/Maniplus (not DEP). Be sure to check the "debug information" box under datamodel properties before working with the debugger. Be sure to set the "checkrules" global setting as appropriate. Breakpoints do not always work.

4.4.1.4 Display()

The DISPLAY command can either wait for user input or is updated dynamically as the script runs. It can be used at critical places to easily show multiple values, though this use has been superseded by the watch window. It can display html.

4.4.1.5 Temporary Files

Temporary files are useful to grabbing information from Blaise database and showing them in grid format to the user in Manipula. Multiple keys defined on a temporary file allow great flexibility in sorting data, and using the INMEMORY setting keeps things fast.

4.4.1.6 Write to a file/MESSAGE()

Customized messages can be written to a physical file to create a log of actions. This is most useful when a lengthy or complicated manipula script is run and the problem occurs intermittently.

Either create your own log file, or use the MESSAGE() command to write to the default error file. A BOI could also be defined to write to a database for better sorting/filtering.

Manipula standard errors include Calcerror, Overflow, RangeError, SubscriptError, ReportImportError. Other settings that influence the file are

MESSAGEFILE = 'FileName' (global setting)

MAXMESSAGE = N (global setting)

Be sure to rename the messagefile in the global setting to something other than "*.msg" so Outlook (Microsoft email program) won't try to open it by default.

4.4.1.7 DLLs

Custom DLLs offers the greatest flexibility of any programming, but require the most work. Manipula has the capability of calling DLLs, which then can do things like create log files/databases, interact with users, read/set system resources, influence interaction with Blaise, and much more.

Be sure to keep to standard programming practices when creating the dll, and release any resources after execution is finished.

4.4.1.8 Late Binding

Manipula scripts generally have needed a datamodel defined before the script can be prepared/run. In 4.8.1.1399, generic scripts can be created that do not need the datamodel. However, all interaction with the datamodel has to be done through manipula datamodel commands.

Take advantage of this and create a library of manipula scripts that handle common situations: load preload data, manage Blaise databases, get counts/status on common fields, and so forth. Debug the scripts once and use them in a variety of situations.

4.4.1.9 Registry Commands

The recent Blaise release has also opened the Windows registry to Manipula. This opens the possibility of transferring data between manipula runs so preferences can be stored and hopefully reduce errors in similar processes.

Warning: be careful, and be sure of what you're doing! Clean up after yourself!

4.4.1.10 Alien Router/Procedure Object

Manipula can now be called from an alien procedure/router call from the datamodel. This allows complex calculations to be placed outside the DEP into a standard library. Update the library, release to projects, and there's no need to prepare the datamodel again.

Note: the language has been extended to allow Manipula to recognize the environment it is called from.

4.5 Cameleon

This scripting language is intended to be used to create data dictionaries for information exported from Manipula. Debugging Cameleon scripts has always been difficult, and the best methods simply have been to insert normal (non-command) text in the output and see what the output looks like.

Like other languages, use consistent indenting when possible. Consider writing new routines in Manipula because of the datamodel information now available.

4.6 Basil

Basil is meant for a self-administered interview. It bridges the gap between Blaise IS and DEP, but has present special difficulties in debugging since all the commands are stored in the text fields of the DEP using pseudo-xml. The layout on the screens is controlled by panels, and position of controls on the screen are all relative to panels.

This means all panel and controls have to be programmed without a syntax checker at design time and errors occur only during runtime.

Develop by making each panel a different color during the design work to make it easy to position controls in a panel, and then use consistent colors when releasing to testers/production.

It's best to only work on a few screens and get them correct, then copy everything as needed. Copy as much as possible and type as little as possible. Make everything consistent.

Basil heavily relies upon Manipula in the background to handle all the processing of commands from the DEP. Use debugging techniques for Manipula.

4.7 Hospital

Hospital hasn't changed much, but a new visualizer interface has been provided.

This utility allows the user to scan through the datafile and examine those things that are a problem. As the cursor moves over the top of each of the items in the datafile the information related to that area is displayed to the right.

The Blaise help states, "This graphical view may help you finding errors and understand the cause of them."

4.8 Control Center (DEP/Cati)

Listed below are the more useful items found within the control center.

4.8.1 Run - Run (DEP)

Some of the more useful parameters:

- Show watch window

- Choose the files to run with (DEP config, menu file, ...)

Surveys often have "gate" variables that prevent from code executing prematurely, and once set it cannot be easily reset (i.e., for household listings, lists, randomization). Reset these by using assignments in menu items, manipula scripts, or "discard all changes" after saving the entry session at critical points.

4.8.2 Database Browser

Allows the user to look through the raw data of a Blaise database, data view, or BOI. Note a new feature of the database browser is that the details view does not show empty fields in the form. Type multi-part keys by using a semicolon between each of the keys, such as 10001;23. Right-click on the tree structure to use the Find feature to quickly locate and select the fields you want to use. Save the view after making your selections (Control Center - <open database browser> - File - Save As).

4.8.3 Structure Browser

The structure browser, like Delta, shows all the information from the datamodel.

Right-click and select details when in the Elements view. Clicking on the top level of the datamodel will bring up the rules for that level, and clicking on any block will bring the rules for that block. You can also drag & drop any field in this view to a Blaise editing window and you can retrieve the full name of the field directly into the code. No typing = reduced errors = less debugging.

When in the Statements view with details, browse to the statement/field you are interested in looking at (sorry, the Find is not enabled here). Click the "Fields involved" tab, then click the details button at the bottom right. This will bring up the same details as in the Elements view within the Statements view, but for blocks allows you to easily browse through the list.

4.8.4 Datafile Management (Rename, Copy, Move, Create, Delete)

Use these tools to manage Blaise databases and avoid the network file copy. You will avoid accidental problems with the databases.

4.8.5 Tools - Environment Options

The Environment Options are important for creating that workspace that leads to better programming.

Select each of these options for a better work environment.

General

Check for changed files

Use tab sheets

Syntax extensions (i.e., put in .bla, .inc, .prc, ...)

Editor

Auto indent mode

Find text at cursor

Create backup

Syntax highlighting (uses the syntax extensions, modify Blaise.sht file)

Show Explorer (double click on includes to open files)

Make use of bookmarks (CTRL-K# where # is 0 to 9 to set a bookmark, then CTRL-Q# to go to the bookmark). Use the Enhanced search and the Explorer for navigating quickly. Always use project files.

4.8.6 Configure Tools

Add commonly used utilities here with shortcuts to reduce time retrieving them.

4.8.7 Internet Workshop, Internet Service Manager

Necessary for creating Blaise IS surveys. (Another paper ...)

4.8.8 Oldeb Db Workshop, Oledb Db Command Builder

Necessary for creating BOI files. (Another paper ...)

4.8.9 Cati Specification, Cati Management

Needed for working with the CATI system. (Another paper ...)

4.8.10 Blaise Data Centre

This GUI interface is superior to the default database browser in that it can export data in text, BDB, or XML formats outside the control center, including internal form keys. It can filter records, group results, create sub databases, compare case data and BOI versioning, and save the export routines as projects. No knowledge of Manipula needed.

The viewing and exporting of data outside Manipula is a tremendous asset. Versioning (with BOI) and comparison of data is especially useful.

4.8.11 Bascula

A utility to do weighting and analysis. Not used in the debugging context.

4.8.12 Menu Editor

The menu editor allows additional debugging tools to the programmer via the menu items. These include

- Invoke DLL Procedure
- Start executable
- Invoke DLL Procedure (Advanced)
- Invoke COM object method
- Start Manipula
- Keyboard action
- Goto field
- Assign field

Depending upon the need, the author could write information to a file, perform a series of assignments, interact with specific testing software, or other needs. Organizations in the past have made use of the menu for specific testing software. A large number of custom functions can be sent as parameters detailing the environment the item was executed.

4.8.13 Diagnostic Tool

This is a wealth of information that is more useful to StatNeth than to the typical programmer. It does show what licenses you have in your installation (Blaise/Components/Internet/CATI/Basil) for registry and file entries.

4.8.14 Blaise Emulator

The emulator has been updated to work in a non-CATI mode. It can generate random data for all fields in a way that is consistent with the rules of the datamodel. Please note that when using these settings (stored as DMD2.teo file in the example below), you will have very active use of the drive where the data is located. This is because it appears that BTEmula flushes the data to the disk after each change.

There will be some paths that are highly unlikely to be reached randomly. You should create data files for these situations, then change the UseScripts and RandomOrder settings, and specify the [DataScripts] section. Run the special cases first, then change your script (set UseScripts=0), then rerun against the same datafile. All the rest of the values will then be filled.

Set properly, this tool has the potential for revealing holes in logic flow that need more careful testing.

```
[EmulatorSettings]
DataFile=DMD2
MetaFile=DMD2
WorkDir=.
BATCHMODE=1
DELAYRANDOMASSIGN=0
MININTERVIEW=0
MAXINTERVIEW=0
MINQUESTTIME=0
MAXQUESTTIME=0
DisableCATI=1
GenerateRandomData=1
GenerateRemarks=1
UseScripts=0
RANDOMORDER=0
```

```
[DataScripts]
Count=3
Script1=DMD2.sc1
Script2=DMD2.sc2
Script3=DMD2.sc3
```

4.9 CATI

The CATI management block now has an auxfield block with history information of the current case before it's written to the database.

The CATI specification program now has a log file of all specification changes. This is very useful to track down when/who made a change to the scheduler.

The CATI management program now has a log file for daybatch exclusions. Unknown users now are identified as <LoginName - ComputerName>.

4.10 Database Monitor

This has been updated to show the name of the machine on which the process is running plus the name of the user that is running the process.

4.11 Blaise IS

Blaise IS pretty much offers the same capabilities as found within Blaise DEP, with a few exceptions, such as manipula scripts, extra DLLs, and so forth are not available on the client side. The same issues as Basil still apply to Blaise IS because all the commands are stored in the text of the survey. Normal tools, such as the Blaise Watch Window, are still present. Extra HTML tags can be used within Blaise IS as normal.

4.12 Modelib

The modelib is used to set the style of pages being shown to the interviewer within the DEP. It interacts via the LAYOUT instructions given in the DEP, and by the default grids and input panes, as well as the configuration file.

It can preview the general layout of each page before it even gets to the point of being tested. Additional information can be found by right-click (other than the entry box) on the page and choose Page/Question properties. This will give information on the grid, info, and question properties for that page.

4.13 Configuration File

The configuration file is nearly the same as the Modelib except that no additional layouts can be created. The configuration can change certain parameters, such as the margins, status bar, and help/audit trails.

It also allows a preview of the layout of the instrument the same way as the modelib.

It is very useful to open datamodel and create a configuration file based upon the datamodel to see what original settings were in the modelib with that datamodel.

A common source of problems for layout/audit/help/status is the interaction between the configuration file and the datamodel. Make sure the configuration file is current for the datamodel. DEP will not warn of an invalid configuration file.

5. Additional Debugging Tools

5.1 Watch Window

The Watch Window has long been a very useful tool for revealing the values of variables as they are being changed. And long has been the mystery of why some values are set, some are skipped, and otherwise unexplained behavior.

The updated watch window has a new option, Show Changed Fields. As a field is entered all other fields that are changed are also displayed. For the example below, the field USE.USE15 has a value entered. It shows that

fields USE.Count_USE_Never, USE.Count_Use_Once, USE.CSE21a, DIWD.DIWDChkPt have all been updated, and the parameters into the procedure GetTimeCodeToSound have also been updated.

This information should be helpful to narrow the possibilities when tracking down a difficult problem.

```
*** General ***
Field changed: USE.USE15 = Once
Blocks checked: 5
Procedures executed: 2
*** Check rules information, 12:37:32 PM ***
Check <main block>
Check USE
> field: USE15 (value: 1)
Check DIWA
> referred field: USE.USE15
> referred field: USE.Count_USE_Never
> referred field: USE.CSE21a
Procedure GetTimeCodeToSound
> parameter: piTimeCode (value: 34)
Procedure GetTimeCodeToSound
> parameter: piTimeCode (value: 10)
> parameter: peSound (value: SOUND (..\MM\430pm.wav))
Check DIWD
> referred field: USE.Count_USE_Never
> referred field: USE.Count_USE_Once
Check DIWE
> referred field: USE.Count_USE_Never
> referred field: USE.CSE21a
> referred field: DIWD.DIWDChkPt
```

5.2 Audit Trail

The audit trail has long been very useful to track down user's interaction with the DEP and determine what series of keystrokes were used to result in a reported problem that comes back to the author and is very useful information.

It is possible to retrieve a backup of the case in XML from the audit.dll, but there are concerns: it is "open" (but so is the audit trail), it creates an additional file, and there is a slight pause as the file is created, and the file extension is .ASC.

The new settings in the .AIF file are

```
BackupRecord=1
AuditFolder=\\DataSrv\Shared\2003
AuditTempFolder=c:\Temp
```

5.3 Alien Routers & Procedures

Alien routers are best used for collecting data in a special format that is not easily obtainable via the DEP, such as using a custom date control. As a debugging tool this has limited value because it is a "real" question that is placed on-route, but will only be processed once it is on-route. The advantage of alien routers as debuggers is that they have access to the data of the datamodel.

Alien procedures are a bit more useful for debugging if there's a real need to dive into the BCP, but will be limited to a few parameters passed in from Blaise into the procedure.

Potentially a great deal of information (access to the entire Blaise datamodel, data, rules checker, ...) can be retrieved and utilized when these alien routines are triggered. However, given that Blaise has more useful means at its disposal (audit trails, xml backup of data, enhanced watch window, calls to Manipula through the Alien call) that writing a custom DLL for debugging purposes may now be more effort than its worth.

6. Creating New Tools

First and foremost, before any new tool is created make sure you cannot do the same task with existing tools. As mentioned before, the Blaise development environment is very rich in details and has plenty of possibilities.

In the past the options for creating new tools were solely using the Blaise Component Pack and interacting with the datamodel. Since the release of Blaise 4.8, the ability to run Manipula scripts from the DEP and the greater flexibility of BOI files has decreased the need of writing .Net applications using the BCP.

Potential tools for use within the Blaise environment include:

- Manipula/Maniplus
- BCP
- Delphi and .Net programming languages
- Other programming languages

I would recommend first seeing if existing tools can be enhanced by using their results, then if Manipula can handle the task, and then finally delve into the BCP as needed.

When creating new tools, always keep object oriented programming practices in mind, and don't be afraid to explore new tools that are constantly appearing, such as the Smart Client Software Factory by Microsoft (<http://msdn.microsoft.com/en-us/library/aa480482.aspx>).

Do your best to have very clear specifications on

- Who are the users
- What is the timeline and budget
- What platform will the utility run on
- What are the expected outputs (reports, results - be very explicit)
- What the interfaces (design and interaction)
- What sort of deployment is expected

Be sure to get all the specifications first, provide an estimate of hours based upon those specifications, and then only start programming once those details are set. Guaranteed, otherwise, the project will not meet the needs of the users, will run out of time or money, will have to be restarted, there will be scope creep, the interface will be poorly designed and unusable, the reports will be insufficient, and so forth.

It's nothing new, but always bears repeating. Spend most of your effort up front in the design and you will save a tremendous amount of effort debugging later.

6.1 Suggested Custom utilities

Custom utilities can always be created are a welcome addition to the suite of tools available to and written by the authors. Some of these may not be exactly a "utility," such as taking advantage of the macro facility within Microsoft Word to perform a complex search & replace operation on source code.

Utilities can be very single-purposed, or general. Below are examples of utilities that an author would find helpful if someone were to write them. Note: these do not necessarily exist except as concepts in this paper.

Utility	Description
Check finder	Locations places in the source code where an inadvertent check was created instead of an assignment. For example, A = 1 is a check, while A := 1 is an assignment. Both are valid syntax.
Fields crosswalk	List the fields/auxfields/locals/parameters that are defined, and how they are used: assigned, ASK, KEEP, SHOW, lookup, SEARCH, ... By examining this list it should be possible to find all unused fields that

	are no longer needed, or old code that no longer serves a purpose.
Layout checker	This would scan the modelib/config file and report any overlaps of panes or fields, or other potential problems where part of one pane/field is overlaid by another.
Completeness checker	Scan the datamodel and report on the fields. This help makes sure each field has been fully defined: Field tag, description, languages, and so forth.
Logic "holes" checker	This may be infeasible per the Turing proof that a program cannot determine if another program will ever run to completion. However, it may be possible to locate some logic holes where a question can never be asked. This is a complex problem, and probably is better revealed by the empirical method via the BTEMula program.
Logic Trace	These are addressed in part by the audit trail, the new "fields changed" watch window option, but a full logic trace may be accomplished by modifying the audit trail dll, or possibly by using an alien procedure to capture every action of Blaise rules checker.
Datamodel Version comparer	Blaise allows versioning of the datamodel, and hence the database. The current database tool, Data Centre, does comparisons between data records on the same database (versioned or non-versioned). The Delta tool performs comparison between two datamodels. However, it may be more useful to have the results in a different format than provided by Delta.

7. Conclusion

Blaise environment is very rich in terms of what it can accomplish for computer-assisted surveys, and as a result is a very complex environment to thoroughly test an instrument.

8. References

<http://www.blaiseusers.org>

Sparks, Peter, "Testing Tips & Tricks," 2007

Sardenberg, Amanda F.; Gloster, John W., "Testing a Production Blaise Computer-Assisted Telephone (CATI) Instrument," 2001, U.S. Census Bureau. Covers the systematic approach to testing an instrument, but not necessarily the tools the author would use to debug an instrument before releasing it for internal testing.

Levinsohn, Jay R.; Rodriguez, Gilbert, "Automated Testing of Blaise Questionnaires," 2001, Research Triangle Institute. Use of keystroke files to verify the same path up to the point being tested.

Sjödin, Sven, "Whatever Happened To Our Data Model? Documenting Change in Continuous Surveys," 2000, National Centre for Social Research, UK. Using TADEQ to document datamodels, and then being able to run comparison between datamodels.

Hofman, Lon; Wings, Hans, "MANIPLUS, A powerful environment for managing Blaise III applications," 1995, Statistics Netherlands, Introduction of Maniplus

Unduplication of Listing Data

Michael K. Mangiapane, U.S. Census Bureau

1. Introduction

The Survey of Construction Listing Instrument (SOC LI) is a CAPI instrument that collects data used to measure new residential home construction in the United States. For this survey, Census Bureau Field Representatives (FRs) visit Building Permit Offices (BPOs) and collect a table of up to 2400 building permits and their associated information. A running pre-sample of the permits is flagged by the instrument to collect extra information such as the location of the residence, the builder of the residence, the owner of the residence, and physical characteristics of the residence. When the listing is completed, the instrument generates a final sample of the permits, and new cases are created from that sample for follow-up using the SOC Questionnaire Instrument (SOC QI), which captures more detailed characteristics of the new residence being built.

Originally a PAPI survey, the instrument was first converted to CAPI in 1997 and used CA Clipper software for the data collection and sampling functions. When the U.S. Census Bureau started to convert its other surveys from CASES or Clipper to Blaise, research was conducted on converting the SOC LI as well. From June 2005 through May 2008 a major initiative was undertaken to convert the SOC LI. It resulted in a Blaise 4.7 instrument that was successfully put into production and is still in use today. One requirement of the instrument is to check a newly entered permit number to make sure it is not a duplicate of any previously entered permit numbers since they are supposed to be a unique identifier. This process is referred to as unduplication. The listing instrument should be able to check for duplicates in a reasonable amount of time, prompt the user that it is a duplicate, and give them the option to correct their entry. Since listing forms can be fairly large it would be relatively easy for an FR to lose track of where they are on a printout and start keying in a permit that was already entered in the case. The Clipper version of the SOC LI was able to perform unduplication immediately after entering a permit number. This functionality was requested for unduplication in the Blaise version since it was something the FR's were used to, and it made sense to check at that point.

1.1 Survey Design

The SOC LI started development in Blaise 4.7.0 b957 and was fielded in Blaise 4.7.1 b1168. It contains a listing table of 2400 lines, each one containing 43 fields. Each permit listed always has a permit number, the date the permit was issued, and the number of housing units. If a permit falls into the pre-sample or final sample, the FR will collect the physical location of the new residence, the builder or owner of the residence, and a respondent category. The FR has the option to collect additional information about a listed permit that falls into sample while they are listing permits during the pre-sampling or after the final sample is run.

The listing instrument also includes two additional "instruments" that were originally written in CASES software. The first is the Questionnaire of Building Permit Officials (QBPO) which collects information about how building permits are recorded in the BPO. The second is the Little Questionnaire Instrument (LQI) which is a subset of the SOC QI and collects physical characteristics of a new residence such as the size of the residence, size of the lot, number of bedrooms, bathrooms, and so on. These instruments are both parallel blocks in the listing instrument and are available as needed.

1.2 Hardware and Software Used

The hardware that the SOC LI currently runs on is a Dell Latitude D400 laptop with an Intel Pentium M processor running at 1.4 GHz and 512 MB of RAM. Microsoft Windows 2000 Service Pack 4 is the primary operating system. As mentioned before, Blaise 4.7.1 b1168 is the version of Blaise currently used in the production version of the SOC LI. All FR's in the field use this software and hardware configuration and all testing times noted in this paper are based on these specifications.

2. Objectives of the Paper

This paper will detail the approaches that were taken to meet the requirements of unduplication, the challenges that were faced within each approach, and the results of prototyping unduplication. The approaches included:

- **Using a procedure** within the listing table that is called after the permit number is entered to compare the current line to all of the lines above it using strings.

- **Using a Maniplus script** that would run after the user indicates they are done listing, using loops to compare a line with all of the lines below it.
- **Using the Blaise API** to call a Visual Basic application that is connected to the instrument via an alien router to perform unduplication using a hybrid of the procedure comparison and the loops of the Maniplus script.

There will also be a discussion of why we decided to use the Blaise API for unduplication and bundling it with a helper program for the SOC LI, the SONC Builder Table. This decision also allowed unduplication to be used in similar listing instruments that were also being converted to Blaise at the same time with minimal changes to the design of the instruments.

3. Unduplication Using a Procedure

One approach for accomplishing unduplication as soon as a permit number is entered is to call a procedure in the rules and let it handle doing the comparison. Since data does not need to be stored from the comparison, and we were aiming to keep the listing rules as simple as possible, it made sense to put the comparison into a procedure. The search itself only looks at permit numbers that are listed on the lines above the newly entered permit number since it is expected that if a number is a duplicate, the incorrect number is the one that was just entered (or the “second” permit number). This also applies even if the FR backs up and edits a permit number. If a duplicate number is found, the FR is prompted with a hard edit check that a duplicate was found and they need to make a correction.

3.1 Procedure Design and Challenges

Since the SOC LI listing table is an array of blocks, one could structure the call to the procedure inside a loop to reach back to previously listed permit numbers. However, that would generate an internal parameter to connect to each line that the procedure has to look at. In this design the instrument would slow down immensely as additional permits are listed since internal parameters are inefficient compared to declared parameters. To work around this we decided to store permit numbers in a string at the table level and pass them into the individual block and then into the procedure. To store 2400 permit numbers of up to 24 characters long and also have the comparison list be comma-delimited, it takes a string of 60,000 characters. Blaise’s current limit for a string is 32,767 characters so two strings are required to hold all of the permit numbers.

After a permit number is entered, a procedure is called to trim any leading or trailing spaces from the current permit number to make the comparison accurate. The unduplication procedure is called and the current line number, current permit number, and the strings of the previously entered permit numbers are passed in. The strings are put into an auxfield that uses the OPEN type so that the comparison only has to be done on one list rather than two separate lists. The length of the first permit number in the list is calculated based on finding the position of the first comma and a loop is started up to do the comparison and step through the list. Each listed permit number in the combined field that comes before the current permit number is checked against the current permit number. If it is not a duplicate, the procedure moves on to the next listed permit number by calculating new start and end positions in the list to pick up the next listed permit number. This continues until the procedure reaches the end of the loop (based on the line number passed in) or until a duplicate is found. The following Blaise code is a snippet of how the unduplication procedure searches for duplicates:

```
temp_Num := Line_Num - 1
temp_PermNum := ''
CombinedField := AllStrings + String2

IF temp_Num = 0 THEN
    BeginPos := 1 {Move on to the next permit}
    EndPos := LEN(CombinedField) {Find the next permit}
ELSE
    BeginPos := 1 {Default to the beginning of the string}
```

```

{Find the end of the first permit number}
EndPos := POSITION(',',CombinedField,BeginPos)

FOR I := 1 TO temp_Num DO
    temp_PermNum := SUBSTRING(CombinedField,BeginPos,EndPos-BeginPos)
    IF Permit_Num = temp_PermNum THEN
        ERROR "@Zs@Z@LA permit number duplicating a previously
            entered permit number has been detected.
            @/Return to the listing and correct the first permit
            number or second permit number@L"
    ENDIF

    BeginPos := EndPos+1 {Move on to the next permit}
    EndPos := POSITION(',',CombinedField,BeginPos) {Find the next
        permit}

ENDDO {I := 1 TO Line_Num}
ENDIF {temp_Num = 1}

```

If a duplicate is found, a hard error is displayed to the FR informing them that the current permit number is a duplicate of a previous entry (See Figure 1).

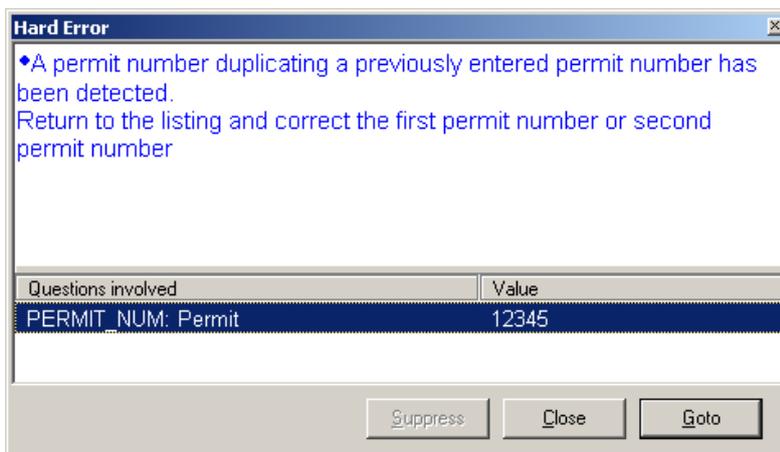


Figure 1: Unduplication error message in the Blaise procedure.

Taking this approach meant that unduplication would run immediately after the permit number was entered. This satisfies the requirement that the FR is prompted as soon as they enter a duplicate permit number. They would be able to correct the duplicate entry and move on without an issue.

3.2 Prototyping and Testing

While testing the SOC LI prototype with this procedure an issue developed as more permits were listed. If a user quit a case before it was complete and then re-entered the case to finish it, the load time to start the DEP would increase depending on the number of permits that had already been listed. Jumping into some parallel blocks and back to the main path of the SOC LI also increased, making it look like the instrument had “frozen.” While this was not noticeable when the number of permits was from 1-200, the lag time became unacceptable for longer listings.

Number of Permits	Instrument Load Time	Table Load Time
100	2 seconds	0 seconds
200	3 seconds	3 seconds
500	12 seconds	10 seconds
1000	56 seconds	54 seconds
2399	9 minutes 40 seconds	9 minutes 46 seconds

It was expected that there would be some lag with longer listings, but we did not expect the lag to become as long as it did within a small number of permits. Surprisingly, unduplication itself was still instantaneous within the table. However, it just took far too long to be able to open up the instrument and get to the listing table and this would have been unacceptable to the FRs. Unduplication using a procedure would be appropriate if the SOC LI was limited to a few hundred permits, but that is not possible with the current design and limits of the survey.

Advantages of using a procedure:

- ❑ Checks for duplicates as permits entered.
- ❑ “Instantaneous” check in listing table.
- ❑ Easy implementation.

Disadvantages of using a procedure:

- ❑ Huge instrument load lag for large listings.
- ❑ Lag introduced when navigating parallel blocks.

4. Unduplication Using a Maniplus Script

Another approach we tried is to take unduplication outside of the instrument and put it into a Maniplus script. Using Maniplus reduces the size of the instrument and makes the instrument more efficient since the instrument does not have to hold a list of the permit numbers and constantly run unduplication on previous lines when the rules are re-executed. Unlike the procedure, the unduplication script checks all of the permit numbers ahead of the permit number that is being examined, though it would be simple to reverse the process to make it more like the procedure.

4.1 Design and Challenges

The main challenge with this approach is that the script had to be associated with an “action” so it would not automatically run after the permit number was entered. Instead, this script ran after the FR exited the listing table via a parallel block. If the FR indicates they are done with listing, the unduplication script is called as one of the actions performed when leaving the parallel block. After determining the last listed line of the table, nested loops are executed. The outer loop is a pointer to the permit number currently being examined, and the inner loop is a pointer that steps through all of the permit numbers below the number in the outer loop to search for duplicates. If no duplicates are found or any duplicate number is marked as “Listed Per HQ”, the instrument moves on to allow the FR to answer some final questions and finish the listing. If a duplicate is found, the script prompts the FR that a duplicate has been found and must be corrected before the listing can be wrapped up as a completed case. The following Blaise code shows the loops inside the Maniplus script and the IF statement that would signal a duplicate permit.

```
FOR I := 1 TO EndPointer DO
  FOR J := I+1 TO EndPointer DO
    IF (Listing.PermiList[I].PERMIT_NUM =
        Listing.PermiList[J].PERMIT_NUM) AND
        (I <> J) AND ((Listing.PermiList[I].PERMIT_REM < 2) OR
                    (Listing.PermiList[J].PERMIT_REM < 2)) THEN
```

This comparison is simple and straightforward enough, but the challenge is in how to display and edit duplicate permits. Information about the duplicates must be displayed to the FRs in an easy to read format so that they can look at each permit and determine which permit number is correct and which one needs to be corrected. Another challenge in this approach is being able to view and correct the permit number quickly without requiring the FR to scroll through several screens to get to the listing line they need. The SOC LI is designed to show 15 permits per screen, if one permit is on line 112 and the other permit is on line 182, they have to press Page Down or Page Up to move ahead or back 4 screens to look at the permits in question. That is a very inefficient way to look at the permits and make the necessary correction.

A solution we developed to overcome these challenges is an approach to unduplication that was created in the National Crime Victimization Survey (NCVS) and detailed in a 2007 IBUC paper on the challenges of converting that survey to Blaise. In this solution, a separate table in the instrument is created that has two lines with the line number, permit number, sample flag, and remark field from the original listing table. The form pane displays the full permit information for both permits in question in a side-by-side format so that the FR can look at the two permits together on one screen to make the comparison (See Figure 2). The unduplication script fills the table with the permit information so that the FR just has make corrections to the displayed permit(s).

Figure 2: Unduplication screen to make corrections.

Once corrections are made, the unduplication script is called again. The script transfers the data from the holding table back into the listing table, and brings the FR back to the wrap-up question for the case. The FR presses Enter and unduplication runs again, repeating this entire process until no more duplicate permits are found in the table.

At the time this approach was not preferred because unduplication did not run until after the listing was done. If there are multiple duplicates an FR has to repeat this search multiple times until all of the duplicates are found, which is less user-friendly than reporting that a permit number is a duplicate right away. This unduplication approach was still a somewhat viable option because it did not slow the instrument down like the procedure.

4.2 Prototyping and Stress Testing

Unlike the procedure-based method, using Maniplus did not impact the load time of the instrument; however, the search for duplicates did take longer as more permits were listed. Since unduplication only runs once at the end of listing rather than multiple times because of rules execution in Blaise, it did not slow the instrument down during listing. We normally run our Maniplus scripts in quiet mode in the instrument, so during unduplication it

looks like the instrument is “frozen” while the script is executing. A message was added advising the FRs that it would take time to check for duplicates (See Figure 3).

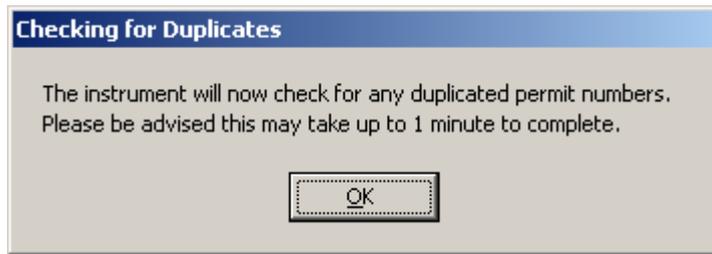


Figure 3: Advisory message to the FRs from the Maniplus script.

It takes less than 1 minute to run unduplication unless all of the listing lines are filled. The following times are based on creating a duplicate permit of the last listed line so that it takes the maximum amount of time to reach a duplicate:

Number of Permits	Unduplication Time
100	2 seconds
200	3 seconds
500	6 seconds
1000	14 seconds
2399	55 seconds

These are better times than what we had observed with using a procedure, satisfying the requirement that unduplication checks for duplicates in a reasonable amount of time. However, using the script meant the instrument did not catch a duplicate permit number as soon as it is entered. This is not what we want from unduplication, but there was a distinct possibility that we would have to use this approach unless another way could be found that was both instantaneous did not slow down the instrument.

Advantages of using a Maniplus script:

- No lag time with instrument load, in listing table, or navigating parallel blocks.
- Fairly easy implementation.

Disadvantages of using a Maniplus script:

- Does not provide the functionality requested – duplicates are not identified until after listing is “complete.” This could lead lots of extra work cleaning up the listing.
- Convoluted way in which FRs had to deal with duplicate permit numbers.
- A one-time lag of up to 1 minute for large listings.

5. Unduplication Using Blaise API to Call a Visual Basic Application

The Maniplus approach showed promise, but had a major drawback. As a result, we decided to see if we could run unduplication outside of the instrument and run it immediately after a permit number was entered. Another application was already being developed in Visual Basic 6 using the Blaise API to work with the SOC LI. We decided to see if an unduplication function could run using Visual Basic and the Blaise API through an alien router without slowing the instrument down.

5.1 Design and Challenges

The unduplication function for Visual Basic 6 takes elements of both the unduplication procedure and the Maniplus script to create a hybrid of each of their approaches to search for duplicates. Since the function runs independently of the instrument via an alien router, it only needs to take the permit number from the current line and use a loop to compare it to all of the permit numbers listed before the current number. The function determines the line number of the current permit number to set the loop counter and stores the value of current permit number. The function enters the loop and checks the current permit number against each permit number entered above it. The Visual Basic code below details the loops and if statements that locate a duplicate permit.

```
If DS.AlienRouterStatus = blrsPostEdit Then
  If ActiveField.Required Then
    For i = 1 To Index - 1
      If WhichSurvey = "SOC" Or (WhichSurvey = "FullView" And
        Mid(db.Field("INTPER").Text, 7, 2) = "SL") Then
        If UCase(Trim(db.Field("Listing.PermitList[" & i &
          "].Undup.PERMIT_NUM").Text)) =
          UCase(Trim(ActiveField.Text)) And _
          db.Field("Listing.PermitList[" & i &
            "].PERMIT_REM").Value < 2 Then
            retval = True ' match found should not be itself
            Exit For
          End If
        End If 'WhichSurvey = "SOC"
      Next i
    End If 'ActiveField.Required
  End If 'DS.AlienRouterStatus = blrsPostEdit
```

The alien router status has to be set to blrsPostEdit, otherwise unduplication runs before the FR enters a permit number. To make sure the permit number comparison is accurate, the two permit numbers are made uppercase during comparison since some permit offices use letters as a part of a permit number. If a duplicate permit number is found, a message is immediately displayed to the FR that a duplicate has been found (and where). Otherwise the cursor just moved on to the next field in the list (See Figure 4).



Figure 4: Unduplication error message from Visual Basic for the SOC LI.

In order to call unduplication from within the instrument, an embedded block is in the listing line. This block contains only the permit number and an alien router to call unduplication.

```
EMBEDDED BLOCK blkUnduplication
  FIELDS
    PERMIT_NUM (PERMIT_NUM)
    ENG "@L?@L.@W[F1]@W
      @/^PermitFill
      @/@LEnter the permit number issued:@L"
    SAS "PERMNUM"
    HLP "H_PNumber"
      / "Permit" : TString24, NODK, NORF

  ROUTER Unduplicate ALIEN('BuilderTableDLL.BT', 'DupEntry')
ENDBLOCK
```

In the field section of the listing line:

```
Undup : blkUnduplication
```

In the rules of the listing line:

```
Undup.Unduplicate
```

Unduplication inside Visual Basic works perfectly in that the FR instantly gets feedback that a permit number is a duplicate. However, an immediate challenge surfaced when this function was implemented. Since unduplication runs as the cursor is leaving the permit number field and runs successfully despite the “duplicate permits” message, Blaise treats that successful run as an indicator that everything is fine so the cursor moves to the next field. The “duplicate permits” message does not come up again unless the FR backs up to the permit number field on that line. If the FR does not back up, then the duplicate permit number is still in the listing. We tried to keep the cursor in place by changing the status of the alien router or even clearing the keyboard buffer in Visual Basic, but it seemed like nothing would work. The solution to this challenge came by doing something most experts say you are not supposed to do when you make assignments to variables in a program:

```
If DS.KeyBuffer = "" = "" Then  
End If
```

Setting the keyboard buffer to empty twice on the same line did not create a syntax error when the function was compiled, but during run-time unduplication would fail and exit immediately without completely finishing the program. Since unduplication exits and returns the result that the program failed, the cursor stays on the permit number field rather than move forward to the next field. A bonus of using this solution is that even if the FR tries to press a function key or click the mouse to leave the permit number field, they cannot leave the field if the permit number is a duplicate, they have to fix it first.

Another challenge that surfaced with this implementation of unduplication was some odd cursor behavior in the instrument when the FR clicked a parallel block tab. If the cursor is on the permit number field, the parallel block tab would briefly have focus, but then the focus would immediately return to the main path and the cursor would go back to the very first question of the SOC LI. If the FR clicks the tab again, the focus goes to the selected parallel block. This does not happen if a parallel block is accessed from a function key, and it only happens if the cursor is on the permit number field. We suspect that it has to do with how the alien router handles focus and have attempted to find a workaround. However, the only workaround at this point would be to remove the tabs from the instrument. This solution would affect functionality in the instrument for the FRs and could be considered worse than the problem it fixes. As a result, it was decided to leave this alone and live with the occasional jump to the beginning of the survey.

5.2 Prototyping and Stress Testing

Since the Visual Basic function is very quick at running the loops and the Blaise API is fast to switch back and forth between unduplication and the DEP, lag is not as noticeable when unduplication is running. When a listing is 1000 permits or more, a lag can be observed. However, this lag occurs only when the user re-enters a previous listing case and enters a new permit number, which runs unduplication for the very first time in that session. The error message or cursor movement is nearly instantaneous after the initial run. All times below are for an initial run of unduplication.

Number of Permits	Unduplication Time
100	<1 second
200	1 second
500	1 second
1000	4 seconds
2399	7 seconds

Advantages of using Blaise API to call a VB application:

- ❑ Very small lag time with first run of unduplication (after reloading instrument).
- ❑ No lag when navigating parallel blocks.
- ❑ Checks for duplicates as permits entered – functionality requested.
- ❑ “Instantaneous” check in listing table (no lag).

Disadvantages of using Blaise API to call a VB application:

- ❑ More challenging implementation – must install DLL on laptops.
- ❑ Focus issue when using the mouse to change parallel blocks from the permit number field.

6. Implementing Unduplication

Based on internal testing of our prototypes, we decided to use Visual Basic and the Blaise API for unduplication in the SOC LI. It does the duplicate check and gives feedback to the FR almost instantaneously so they can correct any problems immediately. It is similar to unduplication in the Clipper version of the SOC LI so the FR does not have to adjust to a new way of being alerted to a duplicate permit. Another program was already being developed for the SOC LI in Visual Basic, the SONC (Surveys of New Construction) Builder Table. Rather than make the instrument keep track of two different files, the unduplication function is coded inside the Builder Table since every laptop that runs a SONC survey has the Builder Table installed as well.

6.1 PAL and NCE

As part of the initiative to convert the SOC LI from Clipper to Blaise, other listing surveys were being converted to Blaise as well. These instruments included the Permit Address Listing (PAL) and Nonresidential Coverage Evaluation (NCE) listing instruments. One aspect of their design is to have them look and feel similar to the SOC LI so it will be easier for an FR to learn how to navigate the instruments. As part of that requirement, unduplication was implemented in PAL and NCE. It makes sense to use the same method for unduplication across all of the surveys since it means minimal changes to the survey design.

Much like the SOC-LI, PAL collects information about newly constructed residential homes in the United States. The major difference between the two surveys is that PAL only collects address information about the new residence and does not sample addresses for follow-up and further data collection. PAL has some extra requirements for unduplication because the sponsoring division wants to allow duplicate permit numbers in the instrument. However, the sponsor also wants to have those permits flagged for review by the FR during listing, or the sponsors during post-processing. Some BPOs for PAL are the same BPOs for the SOC LI, so those cases come to the FR pre-filled with permit numbers from a SOC LI case that had been collected previously. Sometimes permit details change between the time the permit is collected in the SOC LI and when it is being listed in PAL. PAL is designed to keep the old permit from the SOC LI and just add the new one as a line below the old permit. The instrument uses a Maniplus script to insert a line in the listing table and put a flag on that line to indicate that the permit number is a duplicate, and that some other permit information is different from the original. These permits are ignored during unduplication. For brand new PAL cases that do not have permit information from a SOC LI case, the FR is prompted with a message asking if a permit is a duplicate. If they answer yes, the instrument flags that permit line and the instrument moves on. If they answer no, the permit number has to be corrected.

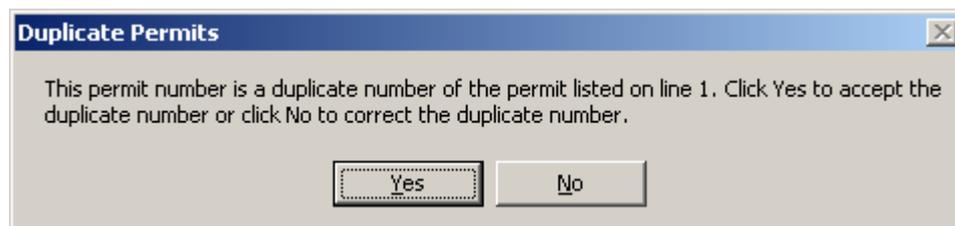


Figure 5: Unduplication error message from Visual Basic for PAL.

NCE is a listing survey that measures new commercial building construction in the United States. The instrument is very similar to the SOC LI so it carries a similar unduplication message. A difference between NCE and SOC LI is that a permit can be flagged for deletion in the NCE instrument. By allowing this flag in NCE, any permit that is flagged for deletion is ignored by unduplication.

7. Conclusion

The SOC LI is a survey that has run continuously in the field for many years and it was our goal during the Blaise conversion that it have a similar look and feel to the Clipper based instrument where possible. Unduplication is extremely important for a survey like the SOC LI because good data quality is absolutely critical in producing data on housing construction in the United States. Unduplication presented a unique challenge in that for a survey as large as the SOC LI it had to run quickly and with minimal impact on the instrument. Thanks to the flexibility of Blaise, there are a number of approaches that can be taken to accomplish unduplication, each with their own unique methods, challenges, and uses inside the instrument. We learned the following lessons about how to use each approach:

- A procedure would be beneficial for a smaller survey instrument that does not need the Blaise API.
- Maniplus scripting would work well if unduplication did not need to be performed immediately after a permit was keyed.
- Using the Blaise API is the best approach for larger instruments that require heavy lifting.

Since deploying the Blaise version of the SOC LI into the field, there have been no required adjustments made to unduplication. In the future we may consider modifying the duplicate permit number search to include any permits entered after the one on the current line. This would cover situations where the FR decides to back up and change a permit number in the middle of their listing.

8. References

Carter, C., Picha, R., Mangiapane, M., Arrington, A., Moshinsky, D.; U.S. Census Bureau, USA (2007): Challenges in Converting the National Crime Victimization Survey to Blaise, Presented at the 11th International Blaise Users Conference 2007, Annapolis, MD, 2007.

9. Acknowledgements

The author would like to acknowledge the following people for input into this paper and input on some of the technical requirements for unduplication.

Erica Filipek, U.S. Census Bureau

Roberto Picha, U.S. Census Bureau

Karen Bagwell, U.S. Census Bureau

Tom Spaulding, U.S. Census Bureau

Malcolm Robert Wallace, U.S. Census Bureau

An End-to-End Solution for Using Unicode with Blaise to Support Any Language

Alerk Amin (CentERdata, Tilburg, The Netherlands)

Richard Boreham (National Center for Social Research, London, England)

Maurice Martens (CentERdata, Tilburg, The Netherlands)

Colin Miceli (National Center for Social Research, London, England)

1. Introduction

Understanding Society is a major new household panel study which has been commissioned by the Economic and Social Research Council (ESRC). Taken as a whole, it will be the largest study of its kind in the world interviewing people in a total of 40,000 households across the UK. It is led by the Institute for Social and Economic Research (ISER) at the University of Essex. The survey will also be known as the UK Household Longitudinal Study (UKHLS) among the academic community, but we will only refer to it as Understanding Society.

The survey will collect data from all members of a household aged 10 and above on an annual basis (10-15s by self-completion only, all those 16+ by CAPI and self-completion). Annual interviewing will allow us to track relatively short term or frequent changes in people lives, and the factors that are associated with them. As the years of the survey build up we will be able to look at longer term outcomes for people in the sample.

There is an ethnic boost in addition to the main general population sample. In the ethnic boost, a household is potentially eligible if anyone (including children) in the household has parents or grandparents from any of these groups: Indian, Pakistani, Bangladeshi, Black-African, Black-Caribbean, Sri Lankan, Chinese, Far Eastern, Middle Eastern and Iranian, Turkish, North African and African Asian.

Response rates among some ethnic groups tend to be lower than the general population, and this can partly be explained by the fact that most surveys exclude non-English speakers, which results in an interviewed population who are not representative of a particular ethnic group.

Providing a translated questionnaire and documents is one way to reduce this bias and to increase response rates by allowing non-English speakers to participate. The questionnaire needs to be translated centrally to ensure that all participants are being asked the same questions in the same way.

In order to determine which languages would be used for translations, we needed to determine the prevalence of people who used that language, but would not have adequate English to participate in an interview. Hence, although Hindi is one of the most widely spoken languages in the UK, it is not one of the languages used in Understanding Society because the majority of Hindi speakers also speak English. The nine languages chosen are

- Arabic,
- Bengali,
- Cantonese,
- Gujarati,
- Punjabi (Gurmukhi script),
- Punjabi (Urdu script),
- Somali.
- Urdu,
- Welsh (there is a legal requirement to provide translations into Welsh even though there are very

few people who speak Welsh who cannot speak English).

These languages include languages with non-Latin scripts (which Blaise cannot deal with) and languages which are written from right to left.

Any interviewer could potentially come across a household where someone speaks one of these nine languages, but where other members of the household can speak English. Therefore we need one questionnaire instrument with English and the other nine languages, and the ability to switch languages during an interview in a single household.

2. Solution Overview

Blaise is the interviewing software used by NatCen so we needed a solution which allowed us to program the main English questionnaire in Blaise. Blaise does not currently support a mixture of Latin and non-Latin scripts in the same instrument, so we needed an alternative to the Blaise DEP which would replicate most of the features of the DEP, but would allow the display of Unicode fonts. So we decided to develop a new data entry program called UNITIP (Unicode Translation Interviewing Program). CentERdata has already previously developed an LMU (Language Management System) for the SHARE survey to manage the translations of the SHARE questionnaire into different languages. We decided to adapt the LMU for use with Understanding Society, by taking the translations stored in the LMU and creating a Blaise questionnaire containing English plus the nine translated languages using the Multi Blaise Generator (MBG) program.

So the two main components to provide the solution to being able to interview in English plus 9 languages are an LMU to manage the translation process and UNITIP data entry program to control the interviewing.

2.1. Translation of Questionnaire

2.1.1. Translation process

There were four stages in the translation process:

- Translation
- Checking by a proof reader
- Checking by independent checker
- Adjudication

The questionnaire was translated online using the LMU using one translator per language. Each translator reviewed their own translation before they code that it is ready for proof reading and they were able to record queries about specific questions in the LMU.

Once the translator had coded that their translation is suitable for proof reading, the agency arranged for a second professional translator to re-read the job. The proof-reader would check the translation against the source text making sure that there are no grammatical mistakes, typos or formatting issues, as well as mistranslations or omissions, including style improvement.

Once the supervisor has coded that the translation is suitable for checking, the translated questionnaire was checked by independent third parties (i.e. checkers). The checkers were either NatCen researchers/freelance researchers, language students/teachers, or qualified linguists. Each language was checked by one checker. They worked by themselves and fed back comments on the translation and the appropriateness of the translation using the 'query' function in the LMU. The checkers checked the translation for accuracy, completeness, interpretation and consistency of meaning with the English questionnaire. All comments and reported errors were sent back to the translation agency for revision.

If the reviewer and translator were unable to reach agreement over the correct translation of an item, then an independent adjudicator resolved the disputed translation.

2.1.2. Questionnaire setup

A type of markup was used in the questionnaire source so elements in the source code could be linked to the LMU via the MBG program.

The first main task was to add a unique tag to each question which required translation. The tags had to be unique to match the unique fields in the LMU, so we added a two character prefix which reflected the block name followed by the question name, for example HH_BIRTH was the birth question in the household grid.

Each element of the question was required to be on separate lines, like this:

```
MvEver (DE_MvEver)
"Have you personally lived at this address your whole life?"
""
/ "LIVED AT ADDRESS WHOLE LIFE" :
TYesNo
```

All standard types which required translation had to be declared at the datamodel level so they could be picked up by the MBG.

We also had some types which didn't require translation and so these were placed in a separate file. This included things like date ranges, numeric ranges and but also questions that could have been translated but we took a decision not to such as country of birth.

Questions which contained a textfill were still a requirement on this project. We had to we had to create a procedure for each question that required a textfill. This technique was used to give the MBG a way to find the textfill in the source code. Here is an example of a procedure before translation:

Note the comment in curly brackets after the assignment of the textfill, this was used as the tag by the MBG to find the rule.

```
PROCEDURE txtLACal
PARAMETERS
  import imLACSx : TBoyGirl
  EXPORT exSFLACal_TFGender2 : STRING
RULES
  IF (imLACSx = A1) THEN
    exSFLACal_TFGender2 := 'he' {SFLACal_TFGender2[1]}
  ELSEIF (imLACSx = A2) THEN
    exSFLACal_TFGender2 := 'she' {SFLACal_TFGender2[2]}
  ENDIF
ENDPROCEDURE
```

And here it is after being processed by the MBG.

```
PROCEDURE txtLChal
PARAMETERS
  IMPORT imLACSx : TBoyGirl
  EXPORT exFTLChal_TFSheHe : STRING
RULES
  IF (imLACSx = A1) THEN
    IF UT.CurrentLanguage = L1 THEN
      exFTLChal_TFSheHe := 'he' {FTLChal_TFSheHe[1]}
    ELSEIF UT.CurrentLanguage = L2 THEN
      exFTLChal_TFSheHe := 'ने' {FTLChal_TFSheHe[1]}
    ELSEIF UT.CurrentLanguage = L3 THEN
      exFTLChal_TFSheHe := " " {FTLChal_TFSheHe[1]}
    ELSEIF UT.CurrentLanguage = L4 THEN
      exFTLChal_TFSheHe := 'e' {FTLChal_TFSheHe[1]}
    ELSEIF UT.CurrentLanguage = L5 THEN
      exFTLChal_TFSheHe := 'ने' {FTLChal_TFSheHe[1]}
    ELSEIF UT.CurrentLanguage = L6 THEN
      exFTLChal_TFSheHe := 'बुच्ची' {FTLChal_TFSheHe[1]}
  ENDIF
```

```

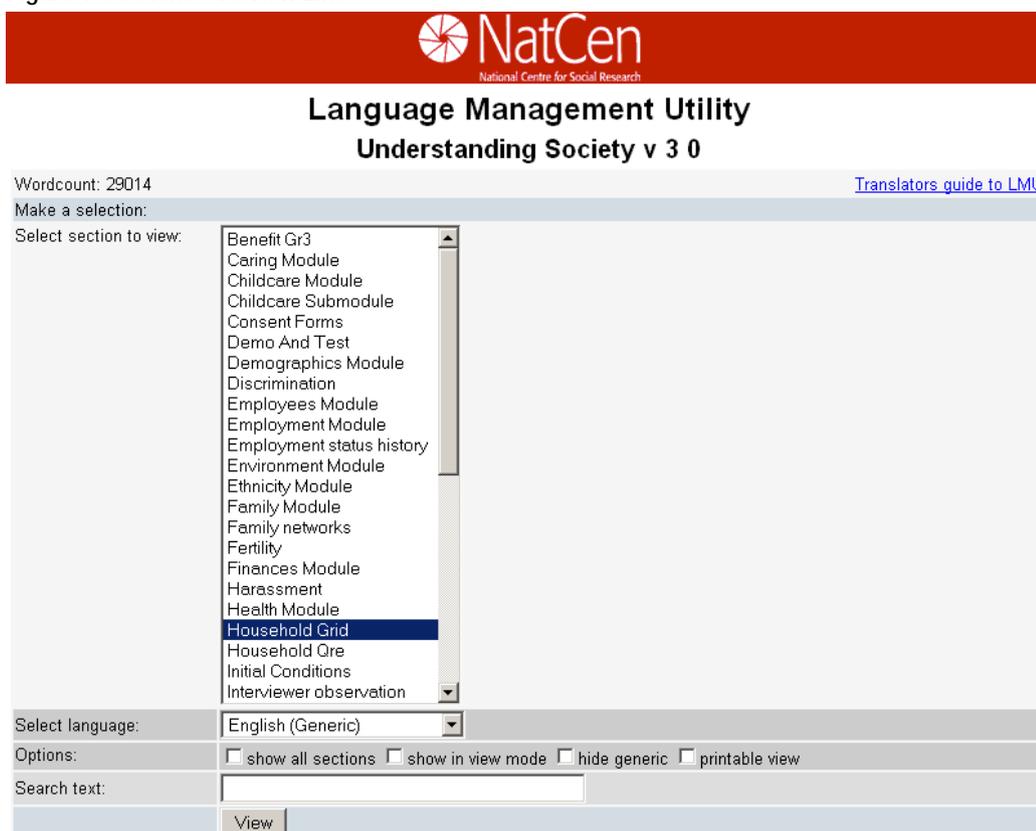
ELSEIF UT.CurrentLanguage = L7 THEN
  exFTLChal_TFSheHe} 'اس لڑکے' =: FTLChal_TFSheHe[1]{
ELSEIF UT.CurrentLanguage = L8 THEN
  exFTLChal_TFSheHe} 'هو' =: FTLChal_TFSheHe[1]{
ELSEIF UT.CurrentLanguage = L9 THEN
  exFTLChal_TFSheHe := '他' {FTLChal_TFSheHe[1]}
ELSEIF UT.CurrentLanguage = L10 THEN
  exFTLChal_TFSheHe := 'ayay' {FTLChal_TFSheHe[1]}
ENDIF
ELSEIF (imLACSX = A2) THEN
  IF UT.CurrentLanguage = L1 THEN
    exFTLChal_TFSheHe := 'she' {FTLChal_TFSheHe[2]}
  ELSEIF UT.CurrentLanguage = L2 THEN
    exFTLChal_TFSheHe := " {FTLChal_TFSheHe[2]}
  ELSEIF UT.CurrentLanguage = L3 THEN
    exFTLChal_TFSheHe := " {FTLChal_TFSheHe[2]}
  ELSEIF UT.CurrentLanguage = L4 THEN
    exFTLChal_TFSheHe := 'hi' {FTLChal_TFSheHe[2]}
  ELSEIF UT.CurrentLanguage = L5 THEN
    exFTLChal_TFSheHe := 'नेप्ली' {FTLChal_TFSheHe[2]}
  ELSEIF UT.CurrentLanguage = L6 THEN
    exFTLChal_TFSheHe := 'ਉਚਦੀ' {FTLChal_TFSheHe[2]}
  ELSEIF UT.CurrentLanguage = L7 THEN
    exFTLChal_TFSheHe} 'اس لڑکی' =: FTLChal_TFSheHe[2]{
  ELSEIF UT.CurrentLanguage = L8 THEN
    exFTLChal_TFSheHe} 'هي' =: FTLChal_TFSheHe[2]{
  ELSEIF UT.CurrentLanguage = L9 THEN
    exFTLChal_TFSheHe := '她' {FTLChal_TFSheHe[2]}
  ELSEIF UT.CurrentLanguage = L10 THEN
    exFTLChal_TFSheHe := 'ayuu' {FTLChal_TFSheHe[2]}
  ENDIF
ELSE
  IF UT.CurrentLanguage = L1 THEN
    exFTLChal_TFSheHe := 'they' {FTLChal_TFSheHe[3]}
  ELSEIF UT.CurrentLanguage = L2 THEN
    exFTLChal_TFSheHe := " {FTLChal_TFSheHe[3]}
  ELSEIF UT.CurrentLanguage = L3 THEN
    exFTLChal_TFSheHe := " {FTLChal_TFSheHe[3]}
  ELSEIF UT.CurrentLanguage = L4 THEN
    exFTLChal_TFSheHe := 'nhw' {FTLChal_TFSheHe[3]}
  ELSEIF UT.CurrentLanguage = L5 THEN
    exFTLChal_TFSheHe := 'नेपो' {FTLChal_TFSheHe[3]}
  ELSEIF UT.CurrentLanguage = L6 THEN
    exFTLChal_TFSheHe := 'ਉਚਨਾਂ ਦੀ' {FTLChal_TFSheHe[3]}
  ELSEIF UT.CurrentLanguage = L7 THEN
    exFTLChal_TFSheHe} 'ان' =: FTLChal_TFSheHe[3]{
  ELSEIF UT.CurrentLanguage = L8 THEN
    exFTLChal_TFSheHe} 'هم' =: FTLChal_TFSheHe[3]{
  ELSEIF UT.CurrentLanguage = L9 THEN
    exFTLChal_TFSheHe := '他們' {FTLChal_TFSheHe[3]}
  ELSEIF UT.CurrentLanguage = L10 THEN
    exFTLChal_TFSheHe := 'ayay' {FTLChal_TFSheHe[3]}
  ENDIF
ENDIF
ENDPROCEDURE

```

2.1.3. LMU/MBG to manage translation process

The Language Management Utility (LMU) is a web-based tool that can be used to translate a specially structured Blaise questionnaire. The questionnaire is divided into several modules. For each of these modules, two separate include files are used. The first file contains a block, with questions and routing. The second file contains the declaration of the fills and the set of procedures. The same modules are also inserted in the LMU.

Figure 1. Sections overview LMU



Every question consists of 7 elements:

- Tag; The tag is used to link questions in the Blaise Questionnaire with their counterparts in the LMU-database
- Name; The first part of every fieldname is uniquely coded by the tag making it easier to group the questions
- Description; Short description of the question text
- Question text; The text the interviewer should read to the respondent
- Help text; The text the interviewer can read for background information on the current question
- Answer; Several types of answers are possible
- Fills; The text variables that are used in the question text to rephrase questions based on certain settings in the environment

The elements are stored in separate fields in the online database. For each language we make the same fields available, but then empty.

The figure below shows the interface that defines a generic question. The elements all are visible in this interface. Also there are options for marking the texts with colors and putting in remarks. This will help us communicate with the translators better. The colors are used to indicate what parts of the question text are new, or what part of a text should not be translated. It is possible to formulate complex answer structures and add fills.

Figure 1. Generic interface for question
LvAg14

The translator has an easier interface. It shows only the texts that needs to be translated. The translator changes the status of a question from 1 to 2 when he is ready.

Figure 3. Interface for the Bengali translator for question LvAg14

Section: Family Module
Language: Bengali (England)
Version: Understanding Society
[main](#) > [section Family Module](#) > LvAg14

LvAg14 (FY LvAg14) Who living with at 14 ?

Generic question:

SHOWCARD F3
Please look at this card and tell me **the number that describes** who you were living with when you were @Baged 14@B.

1. Biological mother and father 2. Adoptive mother and father 3. Mother and stepfather 4. Father and stepmother 5. Mother/no father figure 6. Father/no mother figure 7. In **Local Authority** care/foster home 97. Other

Translation for Bengali (England):

question text

SHOWCARD F3
এই কার্ডটি দেখুন। এটি থেকে আপনার উত্তর বাছাই করে নিয়ে বলুন [@B]১৪ বছর[@B] বয়সে আপনি কার সাথে থাকতেন? শুধু উত্তরের বাম পাশের নম্বরটি বললেই চলবে।

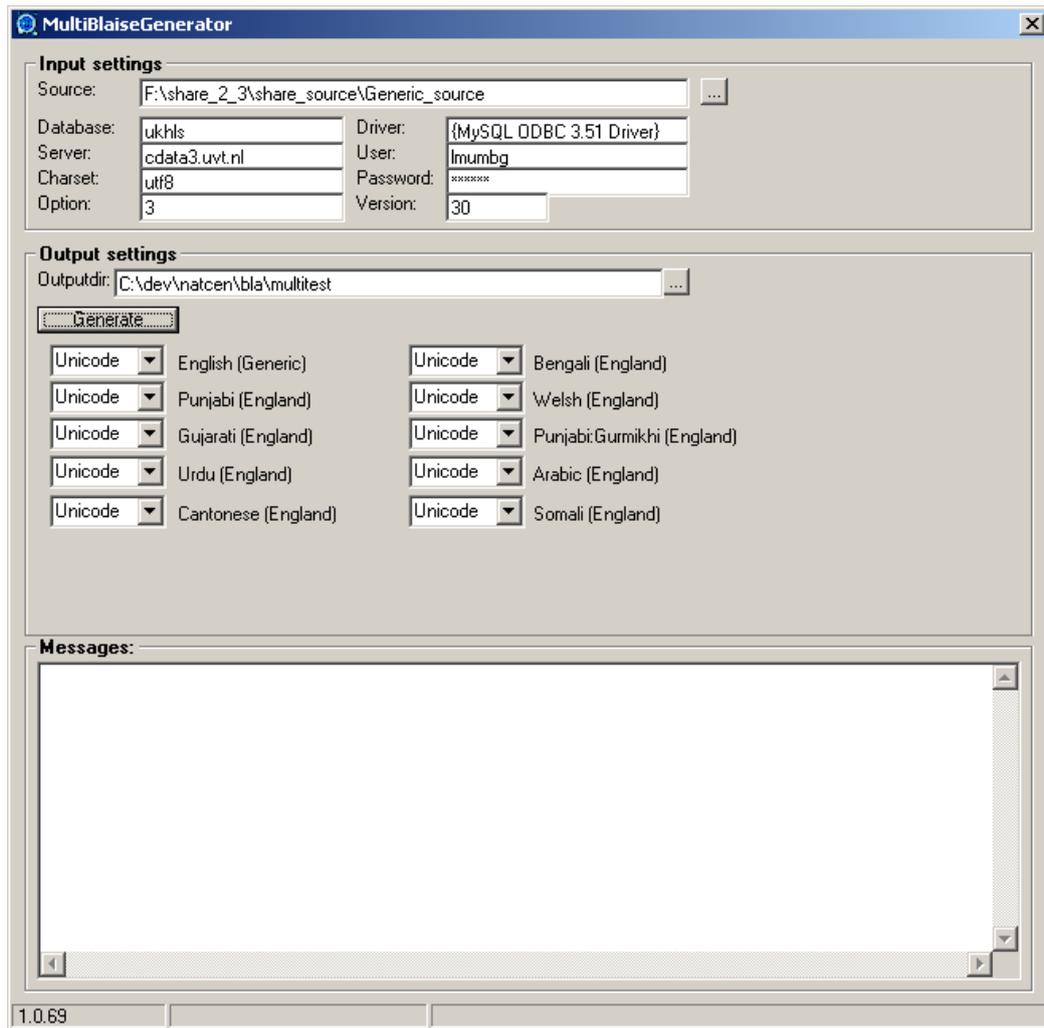
answer type

1.	জন্মদাতা মা ও বাবার সাথে
2.	পালক মা ও বাবার সাথে
3.	মা এবং সং বাবার সাথে
4.	বাবা এবং সং মায়ের সাথে
5.	মায়ের সাথে/বাবা ছিলেন না
6.	বাবার সাথে/মা ছিলেন না
7.	Local Authority-র হেফাজতে/কস্টার হোমে
97.	অন্য কোনো স্থানে

Click to change status: **3.** (3. Supervisor checked translation - ready for reviewer)

When the translation is completed, we run the Multi Blaise Generator (MBG). It takes the elements from the online database and replaces them in the generic questionnaire. This will give us the source code of our original questionnaire, with all the selected languages added to it.

Figure 4. Screenshot of Multi Blaise Generator



2.2. Interview Instrument

2.2.1. Placing Unicode Text Inside a Blaise Questionnaire

Blaise uses an extended ASCII character encoding, with 1 byte per character. This system works fine for alphabets such as Latin, which have a limited number of characters. However, it is insufficient for languages with many characters, such as Arabic or Cantonese.

Unicode is the industry standard for representing text in most of the writing systems in the world. There are several mechanisms for implementing Unicode. Our solution is based on UTF-8, which uses 1-4 bytes for each character. UTF-8 was chosen because it is backwards compatible with ASCII, which makes it compatible with Blaise.

In a Blaise questionnaire, identifiers (such as field names) and text (such as question/answer text) are limited to ASCII characters. In our solution, we keep this restriction for identifiers, so the routing and text fills in Blaise work as normal. However, we replace the ASCII text for questions/answers with UTF-8 strings. This allows us to place multi-byte characters into the Blaise questionnaire, but since UTF-8 is backwards compatible with ASCII, Blaise treats them as ASCII strings.

With this setup, text processing works perfectly. Blaise treats all of the strings as ASCII, so it is able to compile the BLA source file into a BMI, and run the questionnaire. Because the field names are still ASCII, fills and other text processing work properly while running a questionnaire.

Even though text processing works, rendering does not work. Blaise cannot display the UTF-8 strings properly. This affects the usage of the Blaise Editor and the DEP. For the Blaise Editor, any other Unicode-compatible editor can be used, such as Notepad.

For example, the word "No" in Bengali is written as নো, which is a combination of ন (pronounced "na") and া (pronounced "aa"). In UTF-8, the 3-byte code for ন is E0 A6 A9, and the code for া is E0 A6 BE. In ASCII, the codes E0, A6, A9, and BE map to à, |, ", and ¾, respectively. Therefore, the 6-byte code for this word, when rendered in ASCII, looks like "à|à¾". Indeed, this is what the string looks like in the Blaise Editor.

Figure 5. TYesNo answer type, as rendered by the Blaise Editor

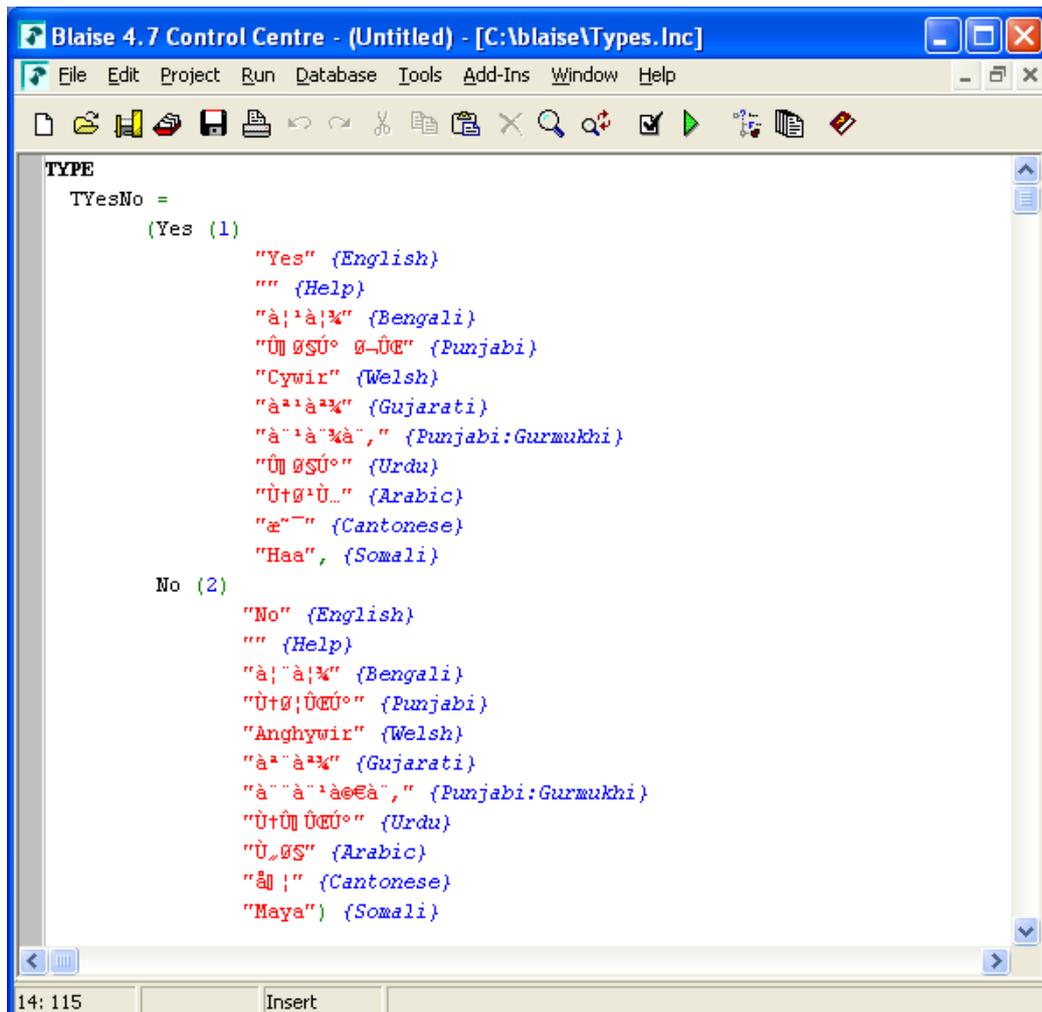
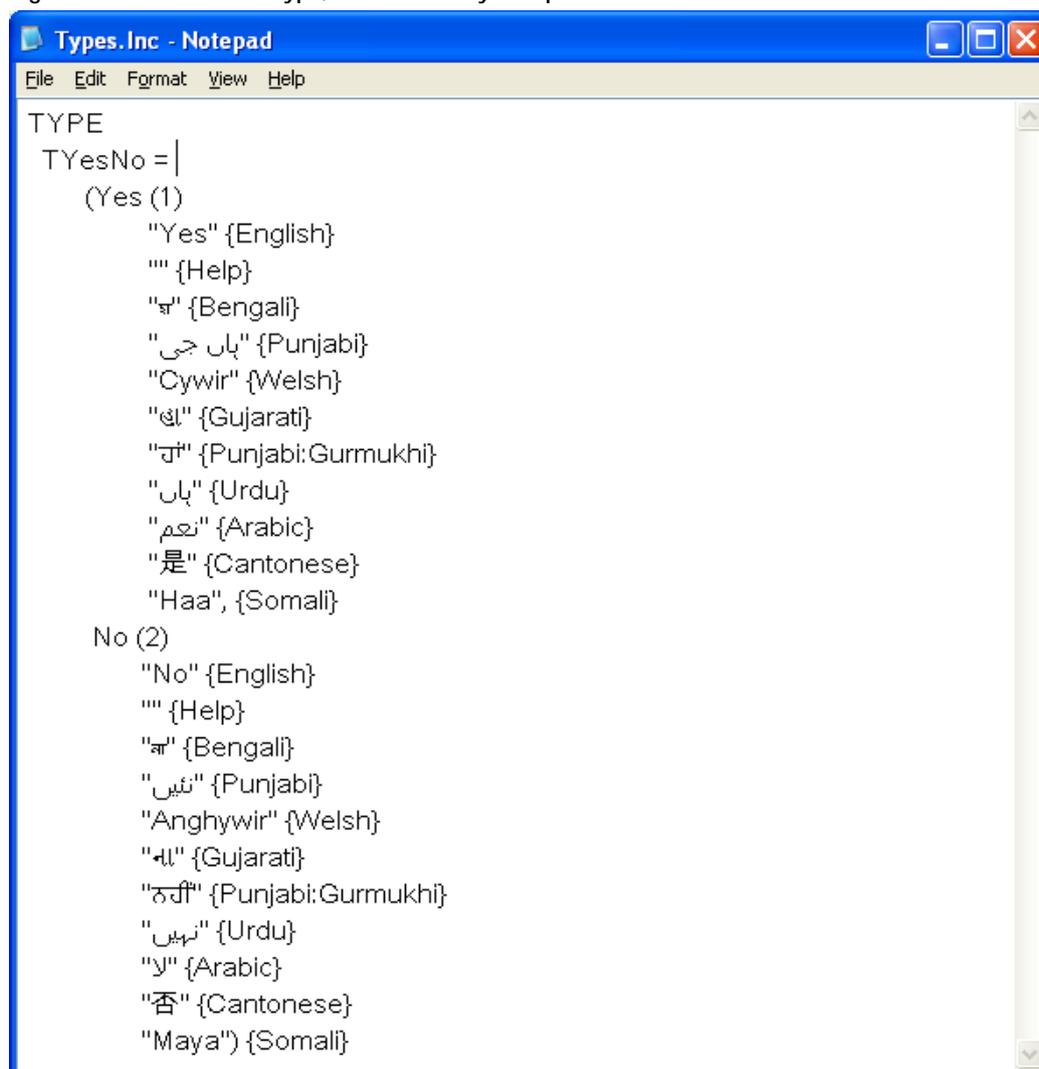


Figure 6. TYesNo answer type, as rendered by Notepad

A screenshot of a Notepad window titled 'Types.Inc - Notepad'. The window contains the following text:

```
TYPE
TYesNo = |
  (Yes (1)
    "Yes" {English}
    "" {Help}
    "হ্যাঁ" {Bengali}
    "ہاں جی" {Punjabi}
    "Cywir" {Welsh}
    "હા" {Gujarati}
    "ਹਾਂ" {Punjabi:Gurmukhi}
    "ہاں" {Urdu}
    "نعم" {Arabic}
    "是" {Cantonese}
    "Haa", {Somali}
  )
  (No (2)
    "No" {English}
    "" {Help}
    "না" {Bengali}
    "نہیں" {Punjabi}
    "Anghywir" {Welsh}
    "ના" {Gujarati}
    "ਨਹੀਂ" {Punjabi:Gurmukhi}
    "نہیں" {Urdu}
    "لا" {Arabic}
    "否" {Cantonese}
    "Maya") {Somali}
  )
```

A Unicode-compatible editor is fine for editing the questionnaire, and creating a BLA file. This file can be compiled with the Blaise Control Center, generating a BMI and BDB. However, when running the file, the Blaise DEP will display the strings as ASCII characters, instead of as Unicode characters. To overcome this program, the UNITIP program was created.

2.2.2. UNITIP

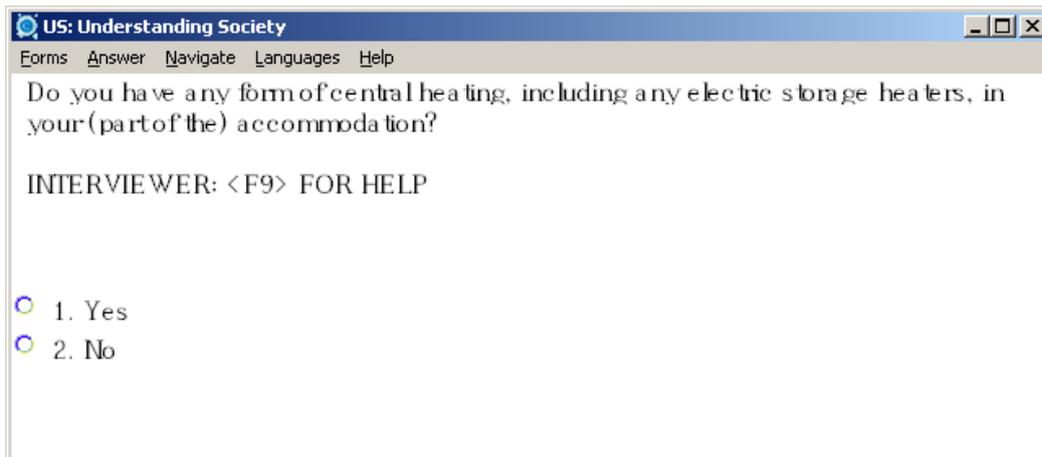
The Unicode Translation Interview Program (UNITIP) was developed to use the Blaise engine to handle questionnaire routing and text processing, but to render Unicode strings.

UNITIP uses the Blaise API to run the questionnaire. All routing and processing is handled by Blaise, to determine which question to display on the screen. Blaise will also handle fills, substituting the appropriate text as needed. While these text fills are really UTF-8 string, Blaise processes them as though they are ASCII. The Blaise API then returns these ASCII strings (that really contain UTF-8 strings) for the question and answer text.

The UNITIP application then renders these strings as UTF-8 strings. It handles left-to-right and right-to-left, as needed.

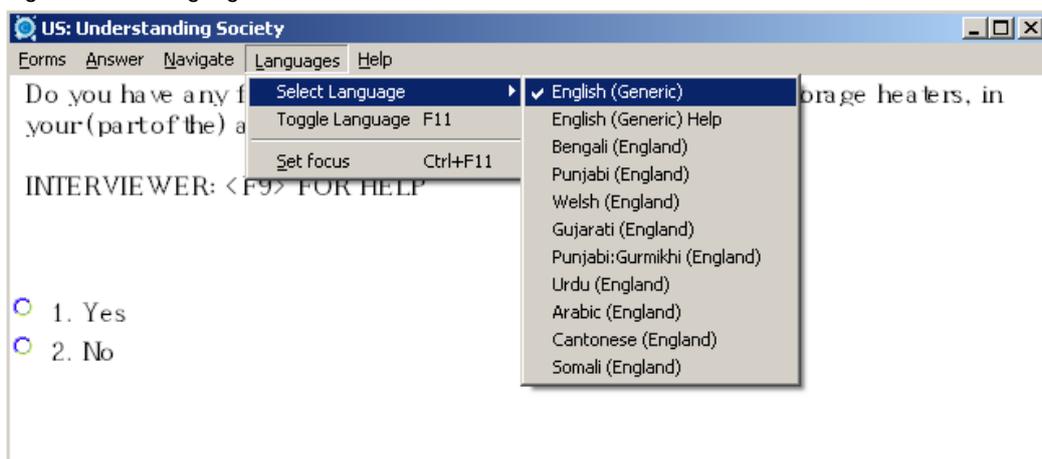
The screenshot below shows the UNITIP program showing a question in English.

Figure 7. An English question in UNITIP



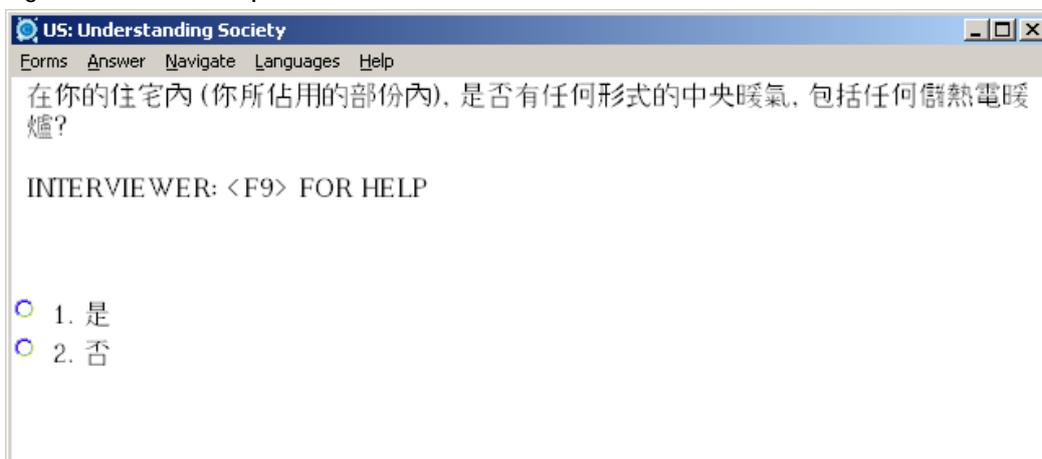
Through the Languages menu, the interviewer can switch to any other language, at any time during the interview.

Figure 8. The Languages menu in UNITIP



If the interviewer changes the language, the view switches to the new language. The following screenshot shows the same question in Cantonese.

Figure 9. A Cantonese question in UNITIP



If the new language is a right-to-left language, UNITIP will change the layout to right-to-left. Additionally, a

different font can be specified for each language. The following screenshot shows the same question in Punjabi, with a right-to-left orientation and a new font.

Figure 10. A Cantonese question in UNITIP



2.2.3. Fonts

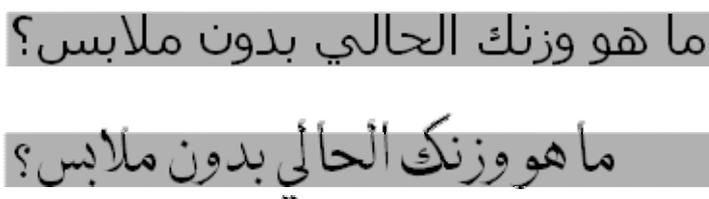
Once everything is set up for displaying a question, the final rendering is ultimately controlled by the font. Choosing a font that is easy for interviewers to read is an extremely important aspect of the questionnaire. However, there are several technical issues that must be considered.

UNITIP can render multiple languages in a single question. This is especially important in the Understanding Society study, as English text is mixed with the other languages for things such as interviewer instructions, names of cities/countries, and more. For each language, a font was chosen that also supports English. For a questionnaire that need to show even more languages in a single question, a font that contains all languages (such as Arial Unicode MS or Code2000) can be used.

The next issue to consider is the size of the font. UNITIP uses information from the Blaise BMI file for hints about the layout of the page. The size of many components, such as Grids, Field Panes, and Info Panes, is based on the size of the default font. The font chosen for each language must be the same height (or smaller) than the default language. This restriction caused a problem for Arabic.

The figure below shows the question "What is your current weight without clothes?" in Arabic. The top is rendered with the Tahoma font, and the bottom is rendered with a Nafees font. In general, Arabic readers prefer the Nafees font, as the varying line thickness and varying heights of characters looks better. However, the gray boxes show the visible area of each font, based on the standard height of the default font. Tahoma fits perfectly, but the Nafees font extends above and below the visible area. These parts of the character will not be visible in UNITIP, thus making the Nafees font unsuitable.

Figure 10. An Arabic question rendered with Tahoma and Nafees



Another issue to consider is bugs within fonts. In Unicode, accents and other diacritic marks are applied to a base characters. In some languages, several diacritics can be applied to a single character. Fonts have layout tables to determine how these should be rendered. We have discovered that some fonts have incorrect layout tables, causing characters to display incorrectly. For example, we discovered that a font used for Bengali was transposing 2 characters, thus rendering as an incorrect word.

Satisfying all of these technical requirements, while also finding fonts that are acceptable for interviewers, is a task that should not be underestimated. There are many considerations to balance, but the result is one of the most important aspects of the solution.

3. Results

3.1. Interviewer Feedback from Translation Pilot

The software was remarkably successful. There had clearly been some glitches but many of these had been resolved. An outstanding issue was the F6 key (used to access more codes if not all appear on one screen) – both the fact that it didn't appear for all questions where it was relevant and that it didn't always work. Despite some freezes or glitches, it was well received by both bilingual interviewers and interviewer-translator pairs. Those undertaking interviews in English seemed to sense no difference from Blaise. Toggling was also clearly very successful – and very extensively used. CAPI translations clearly have the potential to increase the quality of the interview and the responses – and to reduce the time taken by interviews in one of the translated languages.

On the issue of time taken: the timings given by the bilingual interviewers suggested that they were not much longer than interviews undertaken in English. Any extra length in these interviews seemed to be a consequence of

- unfamiliarity with the translated questionnaire
- partly as a result of that extensive (even maybe excessive) toggling
- glossing of questions, where it was felt they were not immediately comprehensible (and sometimes to give a Sylheti version), which was again sometimes linked to toggling to check back what the question was about
- reading out of show cards, where people were not literate in the translated language.

Although there was some feedback that the data entry was not as fast as Blaise, even with the glitches in a pilot version the software was felt to be a considerable improvement on the old paper based system of translations.

3.2. Future Development

Although the overall project went fairly well, we have still some room for improvements. Some parts of the LMU should be made more dynamic. We like to improve our management layer. It should include the option to define user rights, workflow and give a better overview of the current status of the translations. The current version of UNITIP has not yet included the possibility of inserting media, but first experiments show this will not be that difficult.

4. Conclusion

The LMU/MBG/UNITIP solution can help create a multi-language questionnaire, with several languages that, in the past, have been difficult to implement in Blaise. This provides a great number of benefits to the interviewers, interview process, and the data collected for the Understanding Society study.

Managing Translations for Blaise Questionnaires

Maurice Martens, Alerk Amin & Corrie Vis (CentERdata, Tilburg, the Netherlands)

Summary

The Survey of Health, Ageing and Retirement in Europe (SHARE) is a multidisciplinary and cross-national panel database of micro data on health, socio-economic status and social and family networks of more than 30,000 individuals aged 50 or over. SHARE is coordinated centrally at the Mannheim Research Institute for the Economics of Aging (MEA, Germany).

The first wave of data collection was in 2004 (Denmark, Sweden, Austria, France, Germany, Switzerland, Belgium, the Netherlands, Spain, Italy and Greece), Further data have been collected in 2005-2006 in Israel. Two 'new' EU member states - the Czech Republic and Poland - as well as Ireland have joined SHARE in 2006 and participated in the second wave of data collection in 2006-2007. The survey's third wave of data collection, SHARELIFE, has collected detailed retrospective life-histories in sixteen countries in 2008-2009.

The tools discussed in this paper were developed by CentERdata; an independent research institute specialized in data collection and data analysis, housed at the campus of Tilburg University (the Netherlands).

The process of translating the questionnaires and creating the various country-specific versions is aided by several tools. The CAPI instruments are implemented in Blaise and the generic texts (English) of these questionnaires are stored in an external database (MySQL). The different countries translate their version of the instruments using a web application called Language Management Utility (LMU). The questionnaire is broken down into elemental units that can be translated independently. These elements are question texts, interviewer instructions, answer categories, fill texts, error messages and other texts needed for a proper display in the language translated. They are joined into country specific survey instruments, based on the blueprint of the generic version. Translation is done in a cyclic process, making it possible to make changes to the questionnaire during translation. Whenever the generic blueprint changes, the system indicates translation issues with flags. The translated texts are automatically inserted into the Blaise questionnaire using the Blaise Generator application that was also developed by CentERdata.

The tools have also been used to manage the translation process in the UK Household Longitudinal Study, for which a full Unicode supported Interview Program was written (Amin et al., 2009)

1. Introduction

The Survey of Health, Ageing and Retirement in Europe (SHARE) is a multidisciplinary and cross-national panel database of micro data on health, socio-economic status and social and family networks of more than 30,000 individuals aged 50 or over. Interviewers interview household members at their homes with a CAPI questionnaire.

There is a multitude of potential problems with a project of this scale. All countries or even sometimes regions involved have a different fieldwork organization. They have their own systems and workflow. Since SHARE is a panel study some information is forwarded into future waves. Sometimes refresher samples are needed.

The respondents speak different languages and write them with specific character sets, or grammar. They also have different cultural backgrounds, what leads to differences in distinction of manners approaching people. Each wave has a cycle of two years in which the questionnaire is designed, translated, tested and the data are collected.

CentERdata; an independent research institute specialized in data collection and data analysis, housed at the campus of Tilburg University (the Netherlands), was asked to help to program the questionnaires and guide the translation process. In time it became clear that we could also help in automating the complete chain of data-handling. Several tools were programmed; sample management systems, management information tools and tools to transform the data into datasets.

This paper focuses on the ideas behind the methods and tools CentERdata developed to manage the questionnaire programming and the translation process.

First some choices were made: It was decided to use Blaise because of its proved ability for CAPI surveys, the reliable handling of the data, the widely reputation of the program and the picked up knowledge and experience with Blaise at CentERdata. The questionnaire was originally designed in English by teams from various countries. This version is called the generic questionnaire. Afterwards translators had to translate that version. Translators should never make their translations directly in the Blaise source code. Copy- and pasting in translations from a text file into the source code by programmers is also something that should be prevented. Furthermore filling in strings into a question text based on some routing may need different Blaise code for different languages. It would be an ideal task for a system rather than a person.

To make a start we chose to separate each question into elemental translation units and automate the process of joining them back together. Translation is done via a web interface, assuring us the translated text will be in a format we can support and giving us the possibility to stay on top of things.

It should be possible to translate parts of the questionnaire during the questionnaire design hence shortening total development time.

Since some of the languages we had to support need non-Latin character sets we should develop methods to display these languages in the Blaise DEP. The texts were stored on our server in Unicode. We agreed upon a set of basic rules we apply in programming the questionnaire to ensure the translations can be fit into the questionnaire nicely.

Next we will describe the building blocks we used and the constraints we defined on the freedom of programming the Blaise code and how this is displayed in the online translation management tool.

2. Getting the lingual building blocks

Because of the complexity of the questionnaire a very clear structure was chosen.

The questionnaire is composed from several sections with a theme, for instance “income” or “health”. These sections each are defined as a block. Each of these blocks is stored in a separate include file. The blocks can also contain sub blocks still stored in the same file.

Questions and variables all have a name that has a reference to the name of the block. They also have a number, uniquely identifying the question, and a name-part to keep the code readable. These parts together make up the *name* of the question. Furthermore all questions have a *tag*. This tag is unique and can't change. In Figure 1 an overview of one section is given. It shows the structure of question names, tags and how they fit into the section.

Section: ST Starting Section
 Language: English (Ireland)
 Version: Final Version Main
[main](#) > section ST

Questionname	Tag	Description	Action
ST001a_Proxy	(ST001a)	CHECK IF PROXY	 
ST001b_Proxy	(ST001b)	VALIDATE PROXY	 
ST002_Strt	(ST002)	START OF INTERVIEW	 
ST003_name	(ST003)	NAME OF RESPONDENT	 
ST004_namechk	(ST004)	CHECK IF NAME IS CORRECTLY RECORDED	 
ST005_name	(ST005)	NAME OF RESPONDENT	 
ST006_mnthob	(ST006)	MONTH OF BIRTH OF RESPONDENT	 
ST007_yob	(ST007)	YEAR OF BIRTH OF RESPONDENT	 
ST008_dobchk	(ST008)	CHECK IF DATE OF BIRTH IS CORRECTLY RECORDED	 
ST009_mnthob	(ST009)	MONTH OF BIRTH OF RESPONDENT	 
ST010_yob	(ST010)	YEAR OF BIRTH OF RESPONDENT	 
ST011_gender	(ST011)	GENDER OF RESPONDENT	 
ST012_strtcal	(ST012)	START THE CALENDAR	 
ST013_introcal	(ST013)	INTRODUCTION OF THE CALENDAR	 
ST016_proxycheck	(ST016)	PROXY CHECK	 

[View all questions for section ST](#)

Figure 1; overview of section ST

The use of tags is essential because it links the database with translation texts to the questionnaire. All questions consist of these same basic elements. They are depicted in Figure 2.

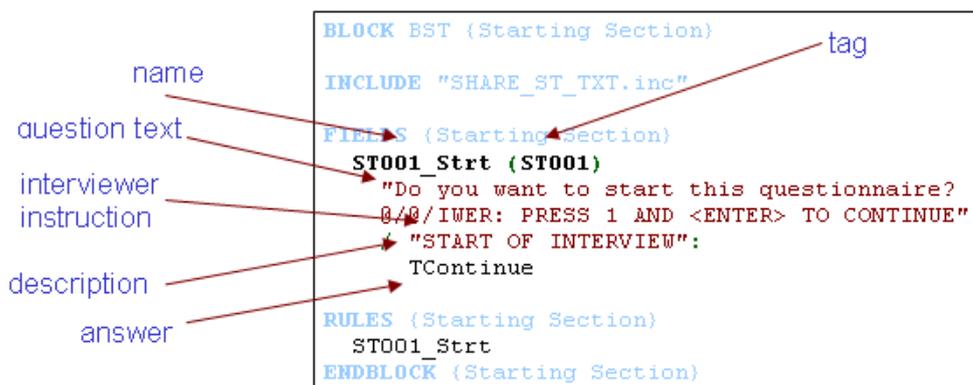


Figure 2; translation elements in Blaise source code

The same building blocks can be found in the online tool:

Section: ST Start questionnaire
Language: English (Generic)
Version:
[main](#) > [section ST](#) > ST001_Stt

name	tag	description
ST001_Stt	ST001	START OF INTERVIEW

Add remark

question text

Do you want to start this questionnaire?

interviewer instruction

PRESS 1 AND <ENTER> TO CONTINUE

answer type

Text Number Memo Range Enum Set, max:

Decimals: TContinue

attributes: (none)

Figure 3; translation elements in the LMU

Each block consists of a set of questions, fills (text variables), a set of procedures for fills and rules. In this context we refer to fields as questions. The *question text* is the text the interviewer should read to the respondent, the *interviewer instruction* is for the interviewer to read, there is a short *description* for each question and finally each question has an *answer type* associated with it.

Answers can be structured in various ways. They can be globally defined answer types or a locally defined enumerated or set type. In general we define answer types global if they are used more than one time.

Some questions are dynamic. This means textual parts of the question change depending on certain environment settings. These variables are called fills and are the solution to many of our language specific problems. We make use of four different types of fills.

- Global fills; are variables that can be used throughout the questionnaire. They include for example birth year
- Answers to closed and open questions;
- Question specific fills; depending on the routing the question should be different in the generic questionnaire
- Translation fills; because of language specific reasons the question text should change based on an environmental setting. For instance based on the gender of the respondent the question should be phrased differently in certain language.

Figure 4 shows the interface of the form a translator should fill in to translate a question.

Section: RP Partner Section
Language: Polish (Poland)
Version: Irish version
[main](#) > [section RP](#) > RP012_prtyrstp

RP012_prtyrstp (RP012) YEAR STOPPED LIVING WITH PARTNER

Generic question:
In which year did you stop living with <code>{FL_RP012_1}</code> ?
IVER:
Translation for Polish (Poland):
question text
W którym roku <code>{fl_rp012_2}</code> <code>{fl_rp012_1}</code> przestaliście mieszkać razem?
interviewer instruction
This question uses the standard answer type 7Year: (1900..2009) → (1900..2009) Click here to translate.
Translate fills for this question:
▶ FL_RP012_1 <code>{name of partner}</code> → <code>{imię partnera}</code>
▶ FL_RP012_2 <code>{empty}{empty}</code> → Pan i Pan/Pani i Pani
Click flag to change status: 
<input type="button" value="Save changes"/>

Figure 4; interface for the translator

On the left-bottom in this interface there are links for fills (FL_RP012_1 and FL_RP012_2). Most names of the fills start with “FL_” then the name of the section and the question number, underscore again and the serial number of the fill itself. Making it easier to identify what question a fill belongs to. The global fills and fills that reference other questions directly differ in this respect. The global fills also start with “FL_” but lack a reference to a specific question (e.g. FL_FirstName). The fills that reference questions use the question name. Clicking both links shows a form where the translation of the fill texts can be entered. “Rules” are shown to give the translator the conditions for the different texts in order to get the same sequences in all languages (Figure 5).

Translate fills for this question:

► [FL_RP012_1](#)

FL_RP012_1
Translate fill text:
Generic fill: 1. (name of partner)
Translated fill: 1. ◀ {imię partnera}
Rules: Is determined by the system

► [FL_RP012_2](#)

FL_RP012_2
Translate fill text:
Generic fill: 1. 2.
Translated fill: 1. ◀ Pan i Pani 2. ◀ Pani i Pan
Rules: IF gender respondent = male THEN [1] {empty} (male text) ELSEIF gender respondent = female THEN [2] {empty} (female text) ENDIF

Figure 5; expanded fill interface

In the language management application all fills that can be used in a question are declared for that questions specifically, even the global ones. Translators are only allowed to use the fills that are defined for the generic question. However, translators don't need to use those fills. If a question can be translated without making use of the available fills they are free to do so. An example of how the translated fills of Figure 5 are loaded into the Blaise source code is shown in Figure 6.

```

FIELDS
FL_RP012_1: STRING[33]
FL_RP012_2: STRING[29]
{ FILL PROCEDURES }

{ STANDARD FILLS SECTION RP }

PROCEDURE Txt_FL_RP012
PARAMETERS
IMPORT piRP004_prtname: TName
IMPORT piW3_CV005_Gender: TMaleFem
EXPORT peFL_RP012_1: STRING
EXPORT peFL_RP012_2: STRING
RULES
peFL_RP012_1 := '' + piRP004_prtname
peFL_RP012_2 := ''

IF piW3_CV005_Gender = a1 THEN
  peFL_RP012_2 := '•Pan i Pan/i' {FL_RP012_2[1]}
ELSEIF piW3_CV005_Gender = a2 THEN
  peFL_RP012_2 := '•Pani i Pan/i' {FL_RP012_2[2]}
ENDIF
ENDPROCEDURE

```

declare fills →

procedure name →

adds space in front of a name →

return texts depend on gender →

fill tags →

Figure 6; example of translated procedure

Note that the translation for FL_RP012_1 is not used in this source code. The code simply adds a hard space in front of the variable that is given to the procedure. The translation for “{name of partner}” is used to make documentation better readable.

There are other texts that are displayed during an interview while using the DEP. They should also be translated but don't need this special structure. We have made them available in the LMU. These include:

- Error texts
- Special global variables like days of the week, name of months, country names.
- Standard types; they can be accessed at question level, but there also is an overview available where all standard types can be seen. See Figure 7.

<u>TAgeGroup:</u>	<ol style="list-style-type: none"> 1. When I was between 0-15 years old. 2. When I was between 16-25 years old. 3. When I was between 26-40 years old. 4. When I was between 41-55 years old. 5. When I was between 56-65 years old. 6. When I was between 66-75 years old. 7. When I was older than 75 years old. 	<ol style="list-style-type: none"> 1. Cuando tenía entre 0-15 años. 2. Cuando tenía entre 16-25 años. 3. Cuando tenía entre 26-40 años. 4. Cuando tenía entre 41-55 años. 5. Cuando tenía entre 56-65 años. 6. Cuando tenía entre 66-75 años. 7. Cuando tenía más de 75 años.
<u>TAgeGroupYouth:</u>	<ol style="list-style-type: none"> 1. When I was between 0-5 years old. 2. When I was between 6-10 years old. 3. When I was between 11-15 years old. 	<ol style="list-style-type: none"> 1. Entre los 0 y los 5 años 2. Entre los 6 y los 10 años 3. Entre los 11 y los 15 años
<u>TAgeIndication:</u>	<ol style="list-style-type: none"> 1. after ^FLEligibleYear 2. (about) ^FLEligibleYear 3. before ^FLEligibleYear 	<ol style="list-style-type: none"> 1. después de ^FLEligibleYear 2. en ^FLEligibleYear 3. antes de ^FLEligibleYear
<u>TAgree:</u>	<ol style="list-style-type: none"> 1. Strongly agree 2. Agree 3. Disagree 4. Strongly disagree 	<ol style="list-style-type: none"> 1. Muy de acuerdo 2. De acuerdo 3. En desacuerdo 4. Muy en desacuerdo
<u>TAmount:</u>	{Amount}	{Cantidad}
<u>TChildren:</u>	{list with children}	{lista con el nombre de los hijos}
<u>TConsent:</u>	<ol style="list-style-type: none"> 1. Consent to preload for this interview 5. No consent to preload for this interview 	<ol style="list-style-type: none"> 1. Da su autorización 5. No da su autorización
<u>TContinue:</u>	1. Continue	1. Continúe

Figure 7; Some standard types in the LMU for a Spanish translation

The LMU can also be used for the labels and texts we use in the tool that work with the DEP:

- Labels for sample management
- A calendar tool was implemented for SHARELIFE
- Help texts are exported to XML

Through the LMU we were able to ‘shield’ all programming tasks from the translation process. It also provided us with tools to monitor the translation process. Each question that is new or has changed is flagged red. The translators change the status of the flag to yellow (meaning: working on, have some question about it or ready for revision) or green (meaning; this question is translated). The flags made it easier to apply changes to an already translated questionnaire; translators would see immediately what questions were changed. A remark field gives them some extra information on why the change was made.

Based on the status of the flags an administrator interface was build that gave insight in the current status of the translation process.

3. Inserting the translation in the questionnaire

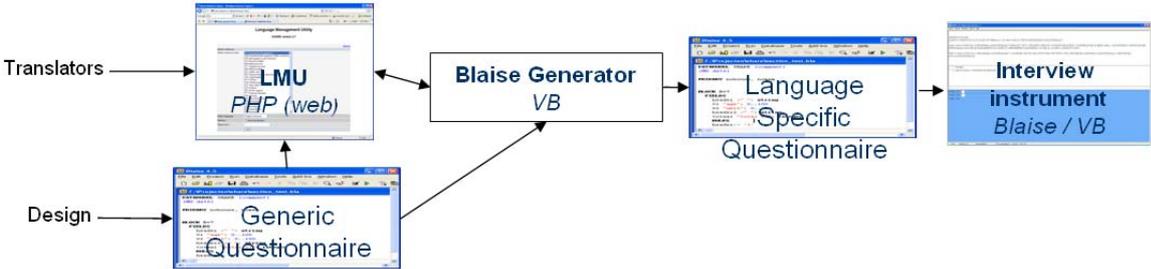


Figure 8; the translation process

To load questions from the LMU into the Generic Blaise Questionnaire a tool named “Blaise Generator” was developed. It takes the building blocks and replaces the texts in a copy of the generic questionnaire. Because some rigid coding constraints were taken into account it can find these texts by simply scanning the text for the tags and replacing the question-elements piece by piece.

It then walks through the fills in the online database and replaces the instances that are defined in the original questionnaire. While doing this, the maximum size of a fill text is determined and the length of the string adapted accordingly.

The global fills and standard types are generated into separate files. The other special translations go to other formats depending on how the tool was written to handle text.

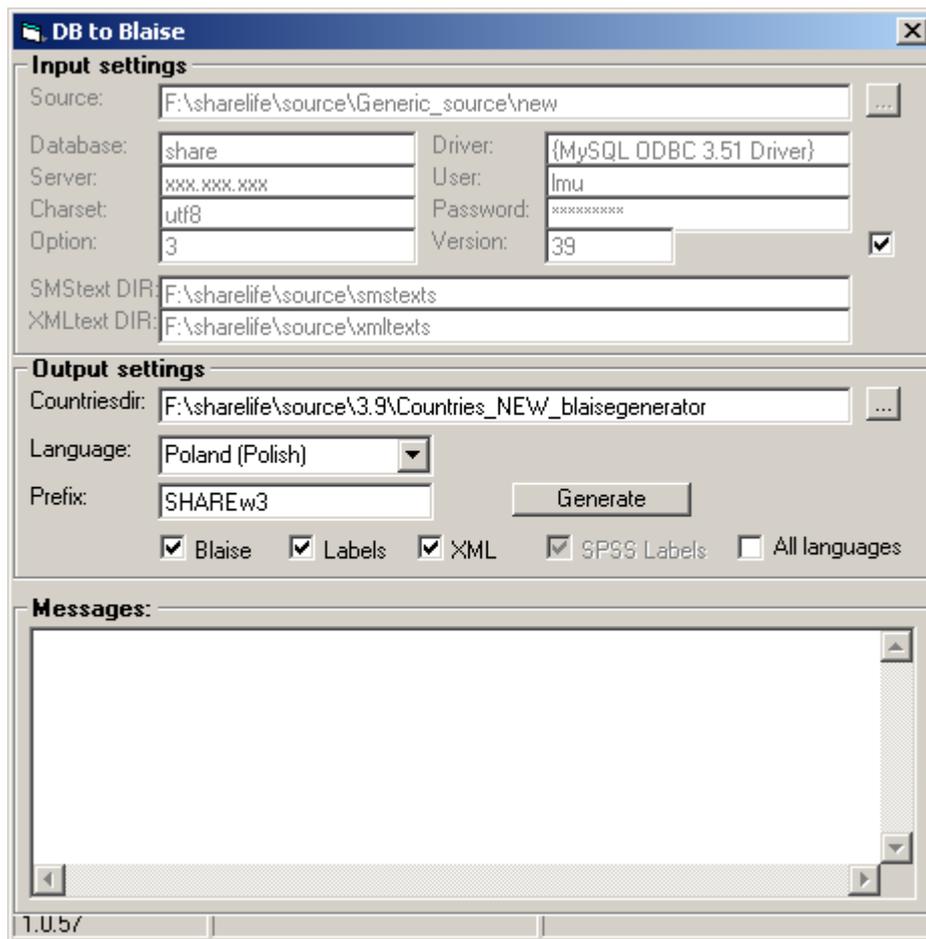


Figure 9; Blaise Generator interface

Since the translations are written in Unicode and stored as UTF-8, we need to convert the texts to display them in the DEP properly. The DEP only displays extended ASCII.

The information and references in the paper Rob Groeneveld presented at IBUC 2003 named “Using non-Latin alphabets in Blaise” was used to determine what format the text in the Blaise code should have.

In general the Blaise Generator has a formula per supported character set of how to translate a given UTF-8 encoded character. For most SHARE-countries this formula is very easy, since the UTF-8 encoding is equal to Latin-1. For Latin-2 (Czech Republic, Poland), Greek (Greece), Hebrew (Israel) and Cyrillic (Israel) the formula could either be subtracting a value from the UTF-code or using a matching table.

For Arab (Israel) the situation is more complex. The way a character displays on screen depends on the position of that character in a word and the surrounding characters. Based on finding in the paper “Using Arabic in Blaise” written by Shani Abel and Shifra Har there was decided to use multiple fonts and let the DEP change fonts when a character is available in another font. The translated questionnaire was analysed and based on the occurrences of characters the fonts were developed, minimizing the number of font switches. The formula for matching the right Unicode character to the right extended ANSI character and the right font was very difficult, but in the end it worked.

Generic question:	
Which is your dominant hand?	
MVER:	
1. Right hand 2. Left hand	
Translation for Arabic (Israel):	
question text	ما هي اليد التي تستخدمها معظم الوقت؟
interviewer instruction	
answer category	
1.	اليد اليمنى. ❌
2.	اليد اليسرى. ❌
Add at position:	3

Figure 10; Arab example in the LMU

Will be placed in the questionnaire as:

```

GS004_DominantHand (GS004)
  ">@k @k@kÅ@kÖiP@k~@k áçÊâ @k"QkëSâE@kî@k'@kE@k ñ@kî@kP@k~@k @óP@k~@k ñê @k"Qkâ"
  /"DOMINANT HAND":
    a1 (1) ".içáóP@k~@k @óP@k~@k .1",
    a2 (2) ".i-³óP@k~@k @óP@k~@k .2"
  
```

Figure 11; Arab text exported into Blaise Source

In Figure 9 the translation is imported into the questionnaire, some characters have @k surrounding them. This means that for this character a different font is used.

The Blaise generator gives us feedback during its process. It indicates whether there are missing tags or fills, and checks for missing answer types. By default it will keep the generic version of a questionnaire element whenever it finds a problem.

When the questionnaire has been pasted in we have a new version of the Blaise source code in another language.

4. Conclusion

The tools that are developed for the SHARE-study have proven to be very valuable. It shortened the total questionnaire development and translation time of these large-scale projects. Furthermore, since we had our building blocks residing in a relational database, we were able to build other tools that generated paper versions of the questionnaire, and it was even possible to build a data-dissemination interface on it.

We have been continually further developing the tools. In future waves of SHARE the complexity of the questionnaire will grow because of changes in the sample, new respondents joined, households split, and respondents die. We remodelled our Sample Management System several times, and every version had some specifics to translate. Feed forwarding of previously gathered information had a huge impact on the structure. In the most recent wave a calendar application was included in the questionnaire. We managed to adapt our tools to these changing circumstances within the project.

The tools have been further developed for the “Understanding Society”-study. This study is run by the National Center for Social Research (London, UK). We had to rebuild our tools to their environment, and leave it for their

people to manage. This resulted in a major update to the LMU regarding the roles that are allowed in a translation process and what phases can be determined. A new program called the “Multi Blaise generator” was built that places multiple languages in one questionnaire. Together with the associated Unitip program it now fully supports all questionnaires in all languages and scripts, making the struggle we had with getting the Arab texts in the questionnaire a nice experiment but obsolete. For the next wave of Share we are a planning to use Unitip.

We plan to keep updating our tools, making them easier manageable by third parties. New interfaces will shorten the support time the IT department has to put in. We think about improving on versioning and porting the translations to an online interview interface.

5. References

Rob Groeneveld, Using non-Latin alphabets in Blaise, 2003

Shani Abel, Shifra Har, Using Arabic in Blaise, 2004

Alerk Amin, Richard Boreham, Maurice Martens, Colin Miceli, An End-to-End Solution for Using Unicode with Blaise to Support Any Language, 2009

Moving to a Centralized Database for Surveys in Blaise at the National Agricultural Statistics Service

Roger Schou, National Agricultural Statistics Service, USA

1. Introduction

The National Agricultural Statistics Service (NASS) is an agency of the United States Department of Agriculture. NASS is responsible for collecting, editing, and summarizing agriculture data. We are the sole agency for producing Agriculture Statistics for the United States. Our headquarters (HQ) is in Washington, D.C. and we also have forty-six field offices, six of which house CATI Data Collection Centers (DCCs). The capacity of the six DCCs range from twenty to sixty workstations. Most of the other field offices also utilize a small staff of interviewers who collect data for their respective offices.

Over the past sixteen years, NASS has conducted surveys in Blaise in forty-four of our field offices. We have always collected and interactively edited data in a distributed environment. The collected data has been stored in Blaise datasets on each field office's LAN. In the late 1990's, NASS opened its first DCC which collected data for several client states. This introduced a complexity of moving zipped up datasets from one location to another. We have now grown to the six DCCs mentioned earlier. Since the data is disseminated, getting a quick snapshot of the progress of a particular survey on a national level is virtually impossible. We also rely heavily on the communication links being up considering how much data is physically transferred from a DCC to a client state.

During the last year, NASS has been thinking along the lines of centralized databases in our enterprise architecture. With the evolution of Blaise 4.81, we are able to pursue storing our Blaise data from all of our field offices in one central database. This will enable us to run national level reports quickly, as well as leverage this architecture to perform our pre and post survey activities. It will also eliminate the need to physically transfer files from one location to another.

2. A Centralized Environment

As we investigate moving Blaise data collection into a centralized environment, we have come to realize that this is not a quick, simple migration. Independently, it would be less involved; however, NASS has decided to move a number of its survey processes to a centralized environment, all within as close a timeframe to each other as possible. This includes centralizing sample master files, the NASS Survey Management System, our in-house web data collection tool (Electronic Data Reporting – EDR), Blaise data collection, our in-house Electronic Data Collection (EDC) tool for smaller surveys, and data analysis tools. Eventually our summary systems will be designed to read these database tables, as well.

What started out as a little splash of an idea to centralize one or two applications at NASS, has turned into a tidal wave on an enterprise level. This change has been welcomed by our agency and senior management has made it a priority. Open communication between all parties involved has been a key. A team has been created to examine Survey Processing in a Centralized Environment. It is made up of representatives from each of the sections involved to brainstorm and plan for this transition. We are trying to coordinate the transitions to the centralized solution so that as many pieces of the puzzle come together at the same time as possible. Many issues have surfaced as these meetings have progressed.

3. Database Decision

One issue that has been a concern is the database in which to store the data. NASS currently makes use of a number of databases including Sybase, Redbrick, FoxPro, Oracle, and MySQL. A proposal has been made to consider MS SQL as an alternative database to some of the existing applications and most of the new development. When centralized, we need as many systems as possible utilizing the same databases. The proposal is currently being reviewed and a decision has yet to be made. In the meantime, we have been instructed to use MySQL as the database for Blaise development, and our CASIC section has based preliminary development of the new system on this directive. A WEB-enabled menu, written in Visual Basic .NET, will be used to run the Blaise portion of the data collection and interactive editing. Blaise 4.81 will be our main tool for collecting and editing the data.

4. Generic BOI Files

We have decided to use Generic BOI files so that there is a common structure across all of the Blaise instruments. By storing the Blaise-collected data in the same eight tables, we envision only one translation tool

to extract the data from the Blaise MySQL database to the Work In Progress database (WIP) which will house the data from all of the data sources (CATI, paper, EDR, and EDC).

5. Versioning

We also plan on activating the Blaise versioning option. One of the requirements proposed by NASS senior management is that the original reported data must be preserved. This has not been the case at NASS in the past. By using versioning in Blaise, we will be able to identify the original data by date and time stamp. This will also allow us to do research on the amount of data that is actually being changed by editing.

6. Folder Structure

NASS has a future vision of opening two large DCCs to handle most if not all of its calling. Until that becomes a reality, we have to provide the ability to run CATI in the six DCCs as well as all of the field offices (FOs). Obviously, the CATI specifications will not be the same in all of the offices. Also, each office will need its own day batch. We have designed a folder structure to handle this challenge. Under each survey folder there will be an FO folder. Under this FO folder there will be individual folders for each field office conducting the survey. In each of these individual folders there will be a CATI specifications file (.BTR), a copy of the .BOI file, and a copy of the instrument files (.BDM, .BMI, and .BXI). We have yet to conduct testing to see if it is absolutely necessary to have a copy of the instrument files in each folder. It would be nice to only have to put them in one physical location.

7. User Access Rights

Another challenge introduced by centralization is user rights. In other words, what data can be seen and/or updated by each user. In the past, distributed instruments and data sets have by default enforced a certain level of user access security due to physical separation. The data has physically existed on different field office servers or in state-level SAS datasets on the Unix box, so the ability of individual field office staff to see data across states has been very difficult, if not impossible. When this data moves to a central database, user access rules will need to be enforced by a combination of database security and logic built into the applications. A preliminary policy has been written, and it has been reviewed and approved by management. A team has been assembled to discuss the details and to prepare for implementing this policy.

8. Additional CASIC Tables

8.1 Information Tables

The CASIC section has introduced three information tables in addition to the generic tables created by Blaise. A Survey Information Table will store needed information about an individual survey such as the folder name, the month and year, the instrument name, the .BOI file name, along with other characteristics of the survey. A Fips Allocation Table will store information such as which states (identified by numeric state FIPS codes) are responsible for the survey, which states are collecting the data, which states are editing the data, the start and end dates which correspond to the data collection period, along with some survey identifier fields. A User Information Table will store information about the users such as their names, employee numbers, and roles. These three tables are in their infancy stage. As plans develop for the other applications it is likely that much of this data will be shared, so the final table design will be determined at a later date. Early thoughts are to create a user access table on the fly from a combination of the information stored in these tables.

8.2 Error Limits Table

Error limits are a necessity for many NASS surveys. We carry limits for things such as acreage and yield for crops, maximum size for grain capacity, litter rates for hogs, maximum number of animals, and size of farm. Our CATI instruments flag warning and critical error messages to interviewers and editors based on these types of limits, which have historically been stored in external Blaise datasets. Error limits in the disseminated scenario have always been survey and state specific, and they have been stored in a Blaise data set unique to each survey in each state. Discussions are leading us to one centralized master error limits table that can be shared by multiple applications. Limits could be added for all states and all surveys. They could be stored, maintained, and accessed in this one location.

8.3 Previously Reported Data

NASS uses previously reported data (PRD) to control routing on subsequent surveys as well as determining an allowable percentage change from previous reports. PRD has been gathered and placed on the sample masters for each survey where it is needed. In a centralized environment, PRD should be accessible directly from the source where they are stored. Time will be freed up for the sampling people to work more with sample design and selecting samples by eliminating the busy work of extracting and posting PRD to the master files. It may be necessary to stage the PRD in a table, as there may be multiple sources for the data.

9. Initializing the Database

NASS maintains a list of farm operators from which survey samples are drawn. Once drawn, the names and addresses are extracted from our list frame. Four files are created from this process which store operator name and address, list frame comments associated with each operation, partners' names and addresses, if any, and sample master data. The current initialize process is a single click of a button that runs a large number of Manipula setups to check these input files, create folders, the external datasets, and ultimately the Blaise dataset.

In the centralized solution, the initialize process will be broken into three parts. The first part is the Survey Setup Process, which will be run by someone in HQ. It will create the folders on the application server and copy both the instrument files and the CATI specification file into each FO folder. The second part is the Initialize Preparation. This step will be run by the field offices. It does the checking of the name and address files against the sample master files. Any problems with the files should be dealt with by each field office, so they will be notified during this process of any issues. Once the files have passed these checks and balances, they will be placed in an area accessible by HQ and an indicator will be set that will signal they are ready for initialize. This brings us to the Initialize step. Most of the time, this will be a CRON job that will execute in the middle of the night to initialize the sample into the MySQL database. There will also be a button on the interface that certain HQ personnel may run for states that miss the deadline for the CRON job. When a state is initialized, another indicator will be set to indicate that they have been initialized.

People in our Survey Administration Branch will be able to keep up with the status of the states' progress during initialize as well as the whole survey process. Reports will be created to quickly show which states have completed the initialize process. This will allow HQ to quickly isolate any problems that may arise.

10. Testing Plans

The CASIC section has developed a testing plan to thoroughly test their system in the new centralized environment. We want to insure that we have not introduced anything that would cause the call scheduler to run less efficiently. In the United States, we must deal with data collection across several time zones. We don't anticipate any problems with the general concepts of the call scheduler, but we have changed the underlying principals to which we are accustomed and so we will plan to give it a thorough testing.

We will also load test the system. We plan on starting with ten interviewers in one location. As long as everything runs smoothly, we will then bring the number up to fifty interviewers in one location. The next step would be eighty interviewers in two locations. We plan on beginning with one instrument and then moving up to four instruments running at one time. The complexity of the instruments will increase as each one is added. On another night, we will have around 200 interviewers in three or four states using one complicated application. We plan to use both the scheduler and Manipula to retrieve cases. We may also want to start several sessions of the BtEmula.exe to get a larger quantity of traffic. Our final test will be during the day with up to 500 users including both interviewers and editors putting as much stress on the system as we might anticipate during our Census of Agriculture. We may also run several sessions of BtEmula.exe during this test phase.

The third test will involve user access. This testing will be done mostly from HQ. We will simply change our roles in the database to achieve different levels of access in order to test rights. Access needs to be restricted based on role and location of the user. For example, someone in the California office should not see data being collected and edited in the Florida office. Another example is an interviewer in one of the DCCs would need access to the state forms for which they are calling, but not forms that are the responsibility of another DCC.

11. State-Developed Blaise Applications

There are many instruments that have been developed by users in our field offices. A state-funded survey is an example of where the CASIC section in HQ would not be responsible for development or maintenance. We have always provided a hook into our HQ menu system for the state-developed Blaise applications. Once the national

surveys have been moved into the centralized solution, we will take a look at the state-developed surveys and explore the possibilities of once again providing a hook into the HQ system.

12. Benefits of Centralizing

We anticipate many benefits of moving to centralized databases for our survey processes. Real-time reporting of survey progress on a national level is something that we have always wished was available. Reports can be written that will be very helpful for the survey administrators. Eliminating the physical transfer of data files from one location to another is a huge step forward. Often, many hours are spent trying to locate data or recreate a transmission of data when a transfer was interrupted. Bringing all of our survey processes together so they may more efficiently communicate with each other will eliminate the “stove-pipe” applications that currently exist in NASS. When systems can be built with a service-oriented approach, the communications between the systems can be seamless.

13. Challenges

There are some challenges that we are proactively pursuing, while others will need to be addressed further down the line. Establishing and implementing a user access rights policy is one of those challenges. Another that is lurking is the coordination of a Blaise datamodel change. What are the steps needed to implement a new datamodel should the definition need to be changed after the start of data collection? What is the best way to communicate to users across the country that the datamodel has changed? How do we coordinate the downing of the CATI Service during data collection, if need be? Other challenges include coordinating multiple modes of data collection in central database. Many of our processes are moving to the centralized environment, but all will not get there at the same time. Dealing with the transition is a big challenge as bridges will need to be built to fill the gaps until all of the processes get centralized.

4. Conclusion

In conclusion, NASS is coming into some very exciting times of development. There is a very positive attitude towards centralizing the sampling, data collection, editing, analysis, and summary processes within the agency. This excitement is shared by the developers all the way up to the senior management. Although progress is slower than originally expected, NASS is making the correct decision by coordinating the transition of all of our survey processes. This will allow the processes to evolve with an inherited communication link between them. Many of the new features offered in the new design and technology can be leveraged when they can first be thought through. In the past, NASS would often place a high priority on turning out a product in a short amount of time and the full potential of a system was never realized. There will always be challenges when moving to a new architecture, but with open lines of communication and cooperation between developers, these challenges will be met.

Blaise Editing Rules + Relational Data Storage = Best of Both Worlds?

Richard Frey and Mike Rhoads, Westat

1. Introduction

Data editing is a critical part of the system lifecycle for survey research projects. Blaise offers flexible and powerful data editing capabilities, and performing editing within the Blaise environment allows us to continue to enforce the rules that were in force during data collection. While the native Blaise data storage format (bdb files) provides an efficient mechanism to support case-by-case editing, it is not as well-suited for supporting aggregate queries that would typically be expressed using SQL.

Since the Blaise Datalink facility now allows Blaise, using OLE DB, to access data in relational databases such as Microsoft SQL Server, it is now possible to use a single data storage format (relational database tables) to support both Blaise-based data editing activities and relational data queries. Blaise 4.8.1 also adds built-in versioning to Datalink, thus providing an off-the-shelf mechanism for tracking all changes made by the data editors. This paper describes our incorporation of these capabilities into a corporate data editing system.

2. Background

2.1 Data Editing Systems

Social science survey research is a process involving a wide range of tasks, including initial design and development, the actual collection of the data, and data delivery and analysis. Some form of post-collection review of the data is essential to ensuring the quality of the data and the reliability of the conclusions that can be drawn from the information. Recognizing this truth, Westat has worked over the years to continually improve the quality of its data editing systems and their integration into the overall project lifecycle (Dulaney and Allan 2001, Gowen and Clark 2007).

Since so much of our data collection is implemented using Blaise software, Blaise has also played a central role in our data editing systems. Blaise offers a number of advantages when used in this manner, such as allowing editors to review and update data either using the same screens seen by the interviewers, or using an alternate layout if that would be more effective for a particular study. The most critical advantage of Blaise, however, is its rules enforcement mechanism. Blaise data collection instruments are designed with range checks, skip patterns, and other checks to enforce consistency of data as they are entered by the interviewers; clearly we do not want this consistency to be lost at the data editing stage. By using Blaise for data editing as well as for data collection, with the original data model used as a starting point, we are able to maintain data consistency without having to reprogram all of the instrument's consistency checks in some other software.

While we have used Blaise as the foundation of our data editing systems, we have typically augmented it with other software, utilizing Manipula or the Blaise API as an interface as necessary. For instance, we have used Microsoft Access for such tasks as managing the status of cases as they move through the editing system and maintaining a data decision log to document all editing decisions. A more fundamental issue for us has been with the native Blaise data storage mechanisms. BDB files work extremely well for storing both simple and relatively complex survey research data, allowing efficient access to cases and the blocks and fields within them for both interviewers and data editors. The underlying storage architecture is not relational, however, and thus is not particularly well-suited for aggregate reporting requirements. In earlier versions of our data editing system, we supported these needs by maintaining parallel data stores in Blaise and in a database, making changes to the Blaise data and then propagating the changes to the database version of the data, either on request or on a scheduled basis. While this proved to be reasonably satisfactory, we still hoped that at some point we could eliminate the need for this redundant data storage.

2.2 Relational Data Storage and Blaise Datalink

2.2.1 The Basics

Fortunately, the Blaise Team at Statistics Netherlands also recognized the value in some circumstances of being able to store data in formats other than native Blaise BDB files, and over the past several versions of Blaise they have significantly enhanced the capabilities of Blaise in this area (Rouschen 2007). One of the biggest steps forward came with Blaise 4.6, when it became possible to use an RDBMS to store not just external data, but the actual questionnaire data itself, using a BOI file (which stores linkage information) in place of a native BDB file.

Another major improvement came with the introduction of “generic” data storage in Blaise 4.8. This supports versioning of the data, which is important in a data editing environment.

Simply put, the Blaise Datalink facility allows the data used by a Blaise data model to actually be stored in a non-Blaise format, such as Microsoft SQL Server, Oracle, or other relational databases. Datalink implements this functionality using Microsoft’s OLE DB (Object Linking and Embedding, Database) technology. This means that it is possible to implement Datalink for any data source for which an OLE DB driver exists. As a practical matter, the type of Datalink system that we required is much easier to implement for a subset of the most common relational data sources, for which Blaise takes into account the characteristics of the particular data source, such as the maximum number of columns it can have in a single table. Blaise does provide such support for SQL Server, which is the database that we are using for our editing system.

Datalink supports four distinct types of BOI files. For our purposes, we use the type of BOI file that is created for an existing Blaise dictionary (data model). This type of BOI file supports not just basic field information, but also field attributes (don’t know, refused), remarks, form information, and block information. It thus can be used as the basis for a data model that supports data editing (or data collection, for that matter), exactly as a BDB file could be used.

2.2.2 Data Partition Types

Once you have decided to use this type of BOI file, you need to determine which of the five “data partition types” you want to implement. The data partition type determines exactly how tables are set up within the relational database to store the Blaise data. We basically used a process of elimination to determine which data partition type to use for the editing system.

The “stream” data partition type stores the data for each Blaise block as a single large column of data in the database—as a binary “blob,” if you will. This maximizes the efficiency of processing by Blaise, since the data for the block is already in the exact binary form that Blaise requires. However, these data blobs can only be decoded by Blaise itself, which did not meet our requirement for a data storage mechanism that could support SQL queries outside of Blaise in addition to within-Blaise access.

Datalink also offers “In Depth” and “In Depth Text” data partition types. Each of these types stores each Blaise field as a row in a database table; “In Depth Text” stores all values in a single text column, while “In Depth” has separate columns for numeric, string, and datetime data. The primary advantage of this storage mechanism is that it minimizes the number of tables and columns required to store the data, while storing each data value in a manner that is somewhat more accessible to non-Blaise software than is the stream type discussed above. Nevertheless, this transposed storage format is still not a good fit for the way most programmers and analysts design report queries. Therefore, we decided that these two data partition types did not meet our requirements particularly well either.

The “Flat No Blocks” data partition type is almost the complete opposite of the “In Depth” partition types. The latter results in a very tall, skinny data store: one table with only a few columns but a large number of rows, since each nonempty field is stored in a separate row of the table. The “Flat No Blocks” type, on the other hand, is an extremely short but wide storage mechanism: there is a distinct column for each iteration of each field (so a block with 5 fields that repeats 20 times would produce 100 columns), and a single row for each Blaise form. In theory, all of these columns would be in a single database table. With all but the smallest data models, however, they are spread out over multiple tables, since databases limit the number of columns that a single table can contain.

The disadvantage of this design for non-Blaise queries and reports is that it is not normalized. Imagine, for instance, that the 5-by-20 block mentioned above is a household enumeration. Locating females in the household would require a query that referenced each of the fields Sex1 through Sex20. Accordingly, we discarded “Flat No Blocks” from consideration as well.

This left us with the “Flat Blocks” data partition type. With this structure, a separate database table is created for every block type in the data model, containing a column for each Blaise field in the block. Each row in the table represents an actual occurrence of the block in the data, e.g. a person enumerated within a household. Each table also contains fields to identify the Blaise form and the instance number of the row within the block.

This data partition type is clearly the closest to the block structure of the Blaise data model, and thus provides the most suitable layout for “natural” SQL queries. For instance, each person enumerated within a household will be a separate row within the appropriate table in the relational database. If we want to obtain some information about all of the females who were enumerated, we can use the single field Sex for filtering, rather than having to use multiple fields as we would with the Flat No Blocks partition type.

For these reasons, we adopted the Flat Blocks data partition type as the basis for the data storage in the latest version of our corporate data editing system. There are some nuances of this data storage arrangement that need to be taken into account when designing queries and reports, and these are discussed later in this paper.

2.2.3 Generic Data Storage and Versioning

With Blaise 4.8, Statistics Netherlands further enhanced its Datalink capabilities by adding support for “generic” data storage. This technique is intended to reduce the burden of RDBMS administration in an environment where there are many surveys (such as a call center) by minimizing the number of tables that need to be created when a new survey is added. This is accomplished by using a set of common database tables that can store data for more than one survey (Blaise data model). Arno Rouschen’s 2007 IBUC paper provides a fuller explanation of generic data storage in Blaise 4.8.

It is important to note that generic data storage is not a “data partition type.” Rather, it is a separate choice that you make when setting up a BOI file to use Datalink. Generic data storage can be utilized with any of the five data partition types discussed above. However, if your goal in choosing generic data storage is to minimize the number of tables in your database (and thus administration burden), some data partition types are much more compatible with this goal than others. In particular, if you choose the In Depth or In Depth Text data partition type along with generic data storage, you can implement new data models under Datalink without requiring any changes whatsoever to the structure of the database that is hosting the surveys.

The Flat Blocks data partition type that we are using for the Westat data editing system does not take full advantage of this capability; new tables must still be created in the database corresponding to each block in every data model. This was not a problem for us, since reducing database administration burden was not a particular concern. In fact, given the large number of different clients for which we carry out data collection, combining data from different surveys within the same database or data tables would create more difficulties for us than it would solve.

The reason that we did decide to utilize generic data storage for our editing system is that it is required to support the versioning capabilities of Datalink, which were also added to the system in Blaise 4.8. When versioning is enabled, Blaise maintains both the old and new versions of records in the database whenever a record is changed, using a special timestamp variable (BEGINSTAMP) to track the history of the record. This process is completely transparent to those who are using DEP or Manipula to access the instrument; they only “see” the latest version of each record. Since all versions are maintained in the database, however, they are accessible to SQL queries, as well as to the new Blaise Data Center tool. In a data editing environment, having the capability to track all changes to the data is obviously attractive.

3. Implementing Datalink for Our Data Editing System

The remainder of this paper discusses how we implemented the Blaise Datalink facility for our corporate editing system. We begin by summarizing the capabilities and characteristics of the system. We then discuss the data storage architecture of previous system versions and show how Datalink greatly simplified our data storage strategy. We describe how Datalink stores the data within the relational database and present some issues to keep in mind when designing SQL queries for the data. We conclude by briefly discussing some tools we found useful when deploying and using the system.

3.1 Editing System Capabilities and Characteristics

As described in more detail in the next section, we incorporated the relational database storage capabilities offered by Blaise Datalink into the latest version of Westat’s Blaise Editing System (BES). This system provides a variety of data editing, reporting and processing options to projects. It includes a set of core capabilities and can be customized to meet the needs of specific project applications. Figure 1 shows the main menu for one implementation of the system.

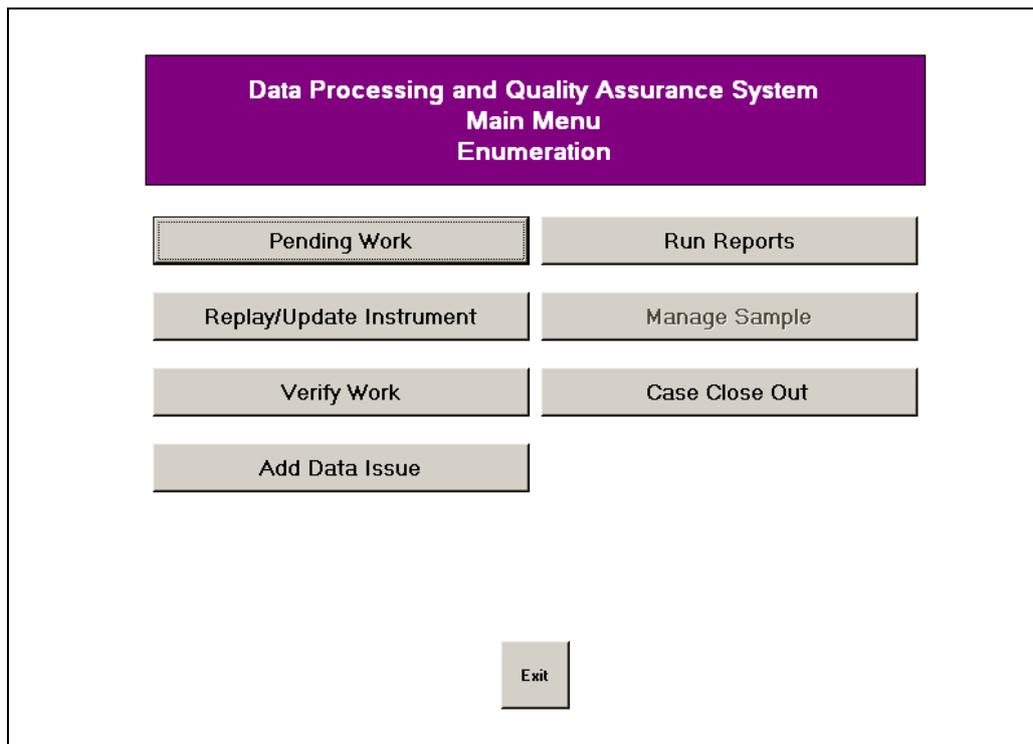


Figure 1. Blaise Editing System main menu

While most of the data processed by the system has been collected using Blaise, this does not have to be the case. For instance, we have used the system to review and edit data that originate in Cardiff's TeleForm software, which we use for paper document capture and processing. Since multi-mode studies may include CAI and hard copy data collection components, this capability allows projects to apply consistent quality control and edit processing regardless of the data collection mode.

In the latest version of the system, all data coming into the system are loaded into SQL Server so that we can take advantage of the Blaise Datalink functionality. The data model used to collect the data (CAPI, CATI or web) is replicated in the home office editing system with the SQL database. Editors review interviewer comments and other specify responses and make updates by replaying the Blaise instrument. Using Blaise for all data changes simplifies the data update process and improves data quality by preventing the unintentional violation of skip patterns during the data cleaning process.

Updates made through interview or form replay are stored in SQL Server, and data decisions are recorded in another database table. The BES decision log module permits projects with requirements to record the reason for data updates to maintain this information in a centralized location while editing is in progress. BES maintains all versions of the data, although only the latest version can be viewed by editors. Audit trails can be activated to recreate prior data history.

BES also provides a variety of reports, covering a wide spectrum of functionality as requested by the project. Reports can present details on workflow monitoring, case edit results, list cases, frequencies and cross-tabs, coding reports, and reports of field comments and data issues. The system integrates with other stages of the project lifecycle by permitting users to view PDFs of codebooks and data dictionaries, and annotated questionnaires or other helpful documentation.

Metadata are maintained using a data dictionary interface. With each new instrument release, the data dictionary is updated with the latest information from the Blaise or Teleform files.

The system normally operates on client computers connected locally to Westat's corporate network. It can also support remote access, with a security portal managing role-based access as appropriate to the user. VB.NET is the primary development tool we used for the system, in addition to Blaise and SQL Server.

3.2 Moving From Many to One – A Simplified Data Storage Architecture

As discussed above, one of the requirements of our editing system was to be able to record and track all changes that were made to data values. This is not an easy thing to do in native Blaise; when a change is made, the previous value is gone. Previously, in order to implement the change tracking capabilities, we wound up using multiple data stores, in BDB and Microsoft Access formats, for each instrument. The data flow for this earlier version of the BES is illustrated in Figure 2.

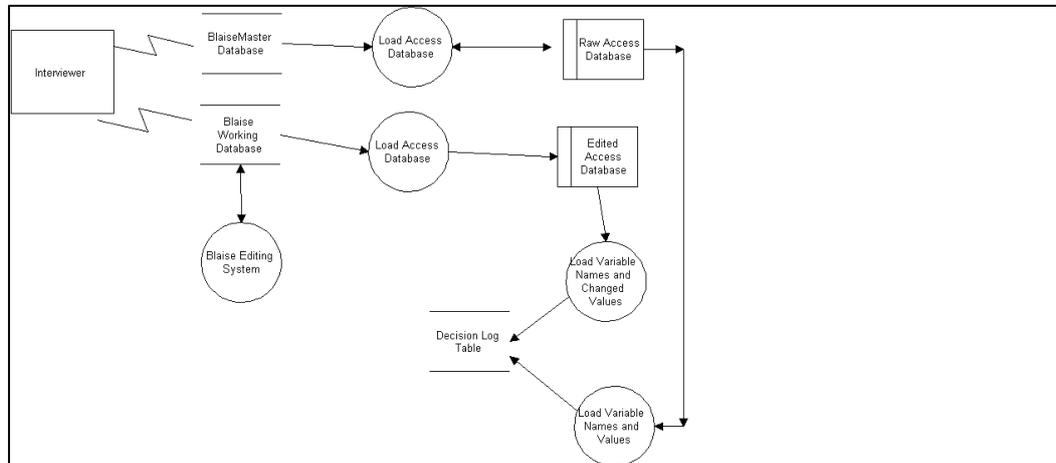


Figure 2. Data flow for previous BES

As data collected by an interviewer arrived for the system to edit (typically in a temporary Blaise BDB file), two separate paths were followed. In order to maintain an easily accessible version of the “original” data (i.e. all values as they were entered by the interviewer), the data were stored in a Blaise Master Database whose values were never changed. From this database, the initial data were also extracted into a Microsoft Access database, and then loaded into the Decision Log Table. The latter is a “vertical” table, with the values of each instrument field being stored as separate rows in the table.

As represented by the lower path on the data flow, the raw data were also stored into a Blaise Working Database. This is the database that was used by the editing application. Once each case was cleaned, the data were extracted into an ASCII file and then stored into the Edited Access Database. Then, all of the changed data from this database were extracted and stored in the decision log table, which is what we used to track data changes.

While this system worked reasonably well, we felt that the architecture could be improved. As the discussion above indicates, the system required a number of different data stores, and data transformations were required between each data store. We used a variety of applications to develop these transformations, including Manipula, Cameleon, the Blaise API, Visual Basic, Microsoft Access, and the SQL Server import/export wizard. Although the transformations were not particularly complex, they did require us to develop and maintain a large volume of code.

Implementing Blaise Datalink with its versioning capabilities allowed us to greatly simplify the design of the system. Rather than having the large number of heterogeneous data stores and storage formats that characterized the previous version of the system, we are able to use SQL Server for all of our data storage needs. We now require a relatively limited number of tables, which can reside within a single SQL Server database.

We did decide to maintain a “vertical table” for change tracking, in addition to the tables that are provided by Blaise Datalink. The Blaise implementation of versioning inserts a complete new set of rows into the database tables whenever any values for a case are changed. This made it difficult to identify the changes in an automated fashion; we found ourselves writing a considerable amount of code to compare rows, dates, and variable values in order to determine which variables and values were changed. Our previously-developed vertical table approach allowed us to store all information about changes in a central location, thus greatly simplifying the task of reporting on data modifications. We streamlined our earlier approach by only storing the variable names and values that changed, rather than all variables. This improvement greatly reduced the number of entries in this table – more than tenfold in some situations, depending on the size of the instrument.

We started to develop Visual Basic modules to keep track of the cases being worked, identifying and extracting all changes from the database tables once updates had been completed. We then realized that triggers and stored procedures within SQL Server would provide a more elegant approach. Knowing that Blaise inserts new rows whenever a data change is made, we created an insert trigger for each database table. When the trigger fires, it calls a stored procedure that compares the values of variables from the “current” row to those in the row representing the previous version of the data. It also takes into account Don’t Know and Refused entries. For each difference that is found, we insert a new row into the vertical table that includes the case ID, variable name, new value, and the login id of the editor who made the change.

```
create trigger [Place Instance Schema Name Here].Place Table Name Here_Update
on [Place Instance Schema Name Here].[Place Table Name Here]
for insert as
begin
if exists (select table_name
          from Information_schema.tables
          where table_name='Place Table Name Here_Inserted' and
                table_schema='Place Instance Schema Name Here')
begin
drop table [Place Instance Schema Name Here].[Place Table Name Here_Inserted]
end
select a.*
into [Place Instance Schema Name Here].[Place Table Name Here_Inserted]
from (select * from inserted) as a

exec [dbo].usp_DPQA_DataUpdates 'Place Instrument Name Here',
                                'Place Table Name Here','Place Key Name Here',
                                'Place Instrument Schema Name Here',
                                'Place Instance Schema Name Here'

end
```

Figure 3. Trigger template

Figure 3 above shows the template that we use to create the triggers for each of the project tables, of which there may be 50-200 or more. As we discuss later in this paper, we created a number of tools to expedite the process of setting a new project up in BES, one of which automated the process of creating these triggers.

3.3 How Datalink Stores the Data

As described earlier, Blaise Datalink offers a number of data storage options (data partition types), and it also allows you to request “generic” storage. The way in which it stores data within your database depends on your choices when you set up your BOI file. For the editing system, we use the Flat Blocks data partition type with the generic data storage option (so that we can take advantage of versioning).

This combination of options results in a “hybrid” storage model, featuring some tables that are specific to the data model and some that are truly generic. In order to query the data outside of Blaise, you need to understand both sets of tables and how they interact.

For the Flat Blocks data partition type, the actual Blaise data are stored in Instrument Tables, which are specific to a Blaise data model. Figure 4 shows the tables for one implementation of the editing system. This implementation contains three data models: WES_CN, WES_EN, and WES_PS. Each table’s name begins with the name of the data model, and there is one table for each block that is defined within the model.

Name	Schema	Created
WES_CN_BBREAKOFF	dbo	4/28/2009
WES_CN_BCN_A	dbo	4/28/2009
WES_CN_BCN_B	dbo	4/28/2009
WES_CN_BPLOAD	dbo	4/28/2009
WES_CN_BTRACE	dbo	4/28/2009
WES_CN_WES_CN	dbo	4/28/2009
WES_EN_BBREAKOFF	dbo	4/28/2009
WES_EN_BPLOAD	dbo	4/28/2009
WES_EN_TENUM	dbo	4/28/2009
WES_EN_TENUM_BENUM	dbo	4/28/2009
WES_EN_WES_EN	dbo	4/28/2009
WES_PS_BBREAKOFF	dbo	4/28/2009
WES_PS_BNCS_AQ5	dbo	4/28/2009
WES_PS_BPLOAD_P5	dbo	4/28/2009
WES_PS_BP5_MOD2SEC1	dbo	4/28/2009
WES_PS_BP5_MOD2SEC2	dbo	4/28/2009
WES_PS_BP5_MODULE1	dbo	4/28/2009
WES_PS_BP5_MODULE2	dbo	4/28/2009
WES_PS_BP5_MODULE3	dbo	4/28/2009
WES_PS_WES_PS	dbo	4/28/2009

Figure 4. Instrument tables

Each of these Instrument Tables contains one column for each non-block field defined within the block, along with a set of 5 special columns that comprise the primary key for the table. JOINKEY and DMKEY uniquely identify a given Blaise case or form, while BEGINSTAMP distinguishes among different versions of the same form. Of course, a block type may be used for more than one field in a data model, and it is common to define an array of blocks. The BLOCKID and INSTANCE fields are used to differentiate these repeating occurrences of block data within a form.

Blaise Datalink, when generic storage has been chosen, also uses a set of common Generic Tables, which are shared among all of the data models in the database. These are shown in Figure 5. Most of them also contain the JOINKEY, DMKEY, and BEGINSTAMP columns, which are the key (pun intended) to joining records from the various instrument tables and generic tables.

Name	Schema	Created
System Tables		
BLAISE_CASE	dbo	4/28/2009
BLAISE_DATA	dbo	4/28/2009
BLAISE_DICTIONARY	dbo	4/28/2009
BLAISE_FORM	dbo	4/28/2009
BLAISE_ID	dbo	4/28/2009
BLAISE_KEY	dbo	4/28/2009
BLAISE_OPEN	dbo	4/28/2009
BLAISE_REMARK	dbo	4/28/2009

Figure 5. Blaise Generic Tables

3.4 Querying the Data

The Flat Blocks data partition stores data in a relatively natural way. For instance, if you have a BJob block in your Commute7Mod data model whose fields include EmpName and Distance, you will wind up with a database table called COMMUTE7MOD_BJOB that will have a corresponding set of columns. In spite of this superficial simplicity, however, there are some important things to keep in mind when querying Datalink data from outside of Blaise.

3.4.1 Distinguishing Current from Historical Records

Remember that when the versioning function of Datalink is turned on, Blaise saves all versions of the data, not just the most recent. While this is desirable when you want to keep track of changes, you have to be sure to filter your SQL queries when you only want to process the latest version of the data.

Figure 6 shows the COMMUTE7MOD_BJOB table records from a simple data model. Note that there are two sets of records for the case with JOINKEY=1 and DMKEY=1, distinguished by their BEGINSTAMP values. The records with the latest values for BEGINSTAMP are current, while the others are historical. (Also note that the EMPNAME for the job with BLOCKID=15 and INSTANCE=1 was changed from CDC to NCHS.)

	JOINKEY	DMKEY	BEGINSTAMP	BLOCKID	INSTANCE	EMPNAME	DISTANCE
1	1	1	2009-04-25 16:46:49.803	4	1	Westat	2
2	1	1	2009-04-25 16:46:49.803	6	2	McDonalds	7
3	1	1	2009-04-25 16:46:49.803	15	1	CDC	12
4	1	1	2009-04-26 13:59:43.520	4	1	Westat	2
5	1	1	2009-04-26 13:59:43.520	6	2	McDonalds	7
6	1	1	2009-04-26 13:59:43.520	15	1	NCHS	12
7	2	1	2009-04-26 13:50:26.370	4	1	Universal Studios	4
8	2	1	2009-04-26 13:50:26.370	6	2	Disney	27
9	2	1	2009-04-26 13:50:26.370	26	1	Waiter on the ...	17

Figure 6. Table with current and historical data

The primary mechanism that Blaise Datalink uses to track record versions is the generic table BLAISE_FORM. Figure 7 below depicts this table for the same datamodel and set of data. You can see that there are two records for the case with JOINKEY=1 and DMKEY=1, indicating that two different versions of that form are represented in the database, while there are only single versions for the cases with JOINKEY values of 2 and 3. Blaise uses a special value of ENDSTAMP, 9999-12-31 00:00:00.000, for rows in this table that correspond to current data. Therefore, the JOINKEY, DMKEY, and BEGINSTAMP values for such rows can be used to identify “current” data rows in any of the other tables. For other rows in BLAISE_FORM, the BEGINSTAMP and ENDSTAMP values can be used if desired to identify the time period during which records with corresponding JOINKEY / DMKEY / BEGINSTAMP values were valid. ENDSTAMP, in other words, effectively acts as an expiration date.

	JOINKEY	DMKEY	BEGINSTAMP	ENDSTAMP	STATUS	COLLECTMODE	DATAENTRYBEHAVIOUR	STREAMDATA
1	1	1	2009-04-25 16:46:49.803	2009-04-26 13:59:43.520	1	-1	3	0xFFFE560045005200530049004...
2	1	1	2009-04-26 13:59:43.520	9999-12-31 00:00:00.000	1	-1	3	0xFFFE560045005200530049004...
3	2	1	2009-04-26 13:50:26.370	9999-12-31 00:00:00.000	1	-1	3	0xFFFE560045005200530049004...
4	3	1	2009-04-26 14:17:29.240	9999-12-31 00:00:00.000	1	-1	3	0xFFFE560045005200530049004...

Figure 7. BLAISE_FORM table

3.4.2 Remarks, Open Text, Don't Know, and Refused

Interviewer remarks and open-text fields are not stored in the Instrument Tables. Instead, they are stored in the generic tables BLAISE_REMARK and BLAISE_OPEN, respectively. Figure 8 shows a sample remark record from our Commute7Mod data model.

	JOINKEY	DMKEY	BEGINSTAMP	FIELDID	REMARKTEXT
1	3	1	2009-04-26 14:17:29.240	9	Not sure how Junior can be 10 years old -- and the only person in the HH

Figure 8. Sample remark record

We see that this table contains a REMARKTEXT column, our usual trio of ID variables (JOINKEY, DMKEY, BEGINSTAMP), and a column called FIELDID. This latter column can be used to identify the field for which the remark was entered, in association with the BLAISE_ID table. This table associates a numeric identifier with each block and field in your data model, making it the equivalent of the Rosetta Stone for understanding how your data are stored. (Note that the creation of this table is optional when creating a new BOI file – be sure to

include it if you want to access your data outside of the Blaise environment!) Figure 9 illustrates a portion of this table.

	DMKEY	ID	TYPE	NAME
1	1	6	F	HHSize
2	1	7	F	Person[1].Name
3	1	8	F	Person[1].Gender
4	1	9	F	Person[1].Age
5	1	10	F	Person[1].MarStat

Figure 9. Selected rows from BLAISE_ID table

As can be seen from the example, the field that is referenced in the BLAISE_REMARK table (ID value of 9) is Person[1].Age. This makes it relatively easy to join these two tables to identify the fields that correspond to each remark.

The text for Blaise fields with type Open are stored in a similar manner, using the BLAISE_OPEN table.

Blaise field attributes (Don't Know, Refused) are also stored in a generic table, rather than in an instrument-specific table. In this case, the BLAISE_DATA table is used, as is shown in Figure 10.

	JOINKEY	DMKEY	BEGINSTAMP	FIELDID	STATUS	STRINGDATA	INTEGERDATA	FLOATDATA	DATETIMedata
1	1	1	2009-04-25 16:46:49.803	46	8	NULL	NULL	NULL	NULL
2	1	1	2009-04-25 16:46:49.803	50	4	NULL	NULL	NULL	NULL
3	1	1	2009-04-26 13:59:43.520	46	8	NULL	NULL	NULL	NULL
4	1	1	2009-04-26 13:59:43.520	50	4	NULL	NULL	NULL	NULL
5	2	1	2009-04-26 13:50:26.370	10	8	NULL	NULL	NULL	NULL

Figure 10. BLAISE_DATA table

Since we are using the Flat Blocks data partition type, the last four columns of this table will always be null. (When one of the In Depth partition types is in effect, these columns are used to store the actual data values.) As with BLAISE_REMARK and BLAISE_OPEN, FIELDID identifies the field in question, while JOINKEY, DMKEY, and BEGINSTAMP identify the case and version. STATUS contains a numeric value corresponding to the field's attribute; Don't Know values are represented by a STATUS value of 4, while Refusals produce a value of 8. Only fields that have a special attribute for a given case have rows in this table.

As with remarks and open text, it is relatively easy to link the BLAISE_DATA and BLAISE_ID tables to identify which Blaise fields have Don't Know or Refused values for a particular case. Unfortunately, if you want to issue a SQL query that shows all of the field values for a particular block, pulling in the attribute information is not nearly as simple. Such columns will have null values in the corresponding instrument table for the block, but we might want to know specifically whether the response was Don't Know or whether it was Refused. Repeating blocks complicate this even more, since each instance of a field will have a separate FIELDID value.

Although cumbersome, the information in the BLAISE_ID table does make it possible to construct such queries. We can take advantage of the fact that the ID values of blocks and fields in the table are assigned in a regular pattern. For instance, for the Marstat field in our BPerson block, we can query the BLAISE_ID table to determine the id values for that field and for its block, as shown in Figure 11.

	DMKEY	ID	TYPE	NAME
1	1	3	B	Person[1]
2	1	14	B	Person[2]
3	1	25	B	Person[3]
4	1	36	B	Person[4]
5	1	47	B	Person[5]
6	1	58	B	Person[6]
7	1	69	B	Person[7]
8	1	80	B	Person[8]
9	1	91	B	Person[9]
10	1	102	B	Person[10]
11	1	10	F	Person[1].MarStat
12	1	46	F	Person[2].MarStat
13	1	82	F	Person[3].MarStat
14	1	118	F	Person[4].MarStat
15	1	154	F	Person[5].MarStat
16	1	190	F	Person[6].MarStat
17	1	226	F	Person[7].MarStat
18	1	262	F	Person[8].MarStat
19	1	298	F	Person[9].MarStat
20	1	334	F	Person[10].MarStat

Figure 11. ID values for Person and MarStat

Noting that the ID values for the instances of the Person block field and its MarStat field repeat in regular patterns (11 and 36 respectively), we can construct a query like the following that displays selected fields from this table for the current version of all cases.

```

SELECT
  BP.JOINKEY, INSTANCE, NAME, GENDER, AGE,
  CASE
    WHEN MARSTAT = 1 THEN 'Yes'
    WHEN MARSTAT = 2 THEN 'No'
    WHEN MSATT.STATUS = 4 THEN 'DK'
    WHEN MSATT.STATUS = 8 THEN 'REF'
    ELSE 'INAPP'
  END AS MARSTAT_TEXT
FROM
  dbo.BLAISE_FORM AS BF
  INNER JOIN dbo.COMMUTE7MOD_BPERSON AS BP
  ON BF.ENDSTAMP = '9999-12-31 00:00:00.000'
  AND BF.JOINKEY = BP.JOINKEY
  AND BF.DMKEY = BP.DMKEY
  AND BF.BEGINSTAMP = BP.BEGINSTAMP
  LEFT JOIN dbo.BLAISE_DATA AS MSATT
  ON BP.JOINKEY = MSATT.JOINKEY
  AND BP.DMKEY = MSATT.DMKEY
  AND BP.BEGINSTAMP = MSATT.BEGINSTAMP
  AND MSATT.FIELDID BETWEEN 10 AND 334
  AND (BP.BLOCKID * 36) + 2 = MSATT.FIELDID * 11
ORDER BY
  BP.JOINKEY, INSTANCE

```

Clearly this is cumbersome – one would not want to code this by hand very often, if ever. Computers, fortunately, are good at such menial tasks, so you could certainly automate the process of developing a set of

queries or database views to meet your needs. These would require joining a separate instance of BLAISE_DATA for each field that allows Don't Know or Refused values, so performance would have to be evaluated.

3.4.3 Linking Up Associated Records

When you define an array of blocks in your data model, the INSTANCE variable in the corresponding Instrument Table can be used to identify the array element for a given row in the table. Thus, in our Commute7Mod data model, where we have an array of persons within a household, INSTANCE is equivalent to a person-number field. When arrays are nested, however, INSTANCE only works at the innermost level. For example, if we can have multiple jobs for each person in a household, INSTANCE identifies the number of the job for the person in question, but there is no direct way of identifying which person holds the job. Here again we can use the information in the BLAISE_ID table (see Figure 12 below) to derive a higher-level identifier should we need to do so – for instance, to join person-level and job-level items.

	DMKEY	ID	TYPE	NAME
1	1	3	B	Person[1]
2	1	4	B	Person[1].Job[1]
3	1	6	B	Person[1].Job[2]
4	1	8	B	Person[1].Job[3]
5	1	10	B	Person[1].Job[4]
6	1	12	B	Person[1].Job[5]
7	1	14	B	Person[2]
8	1	15	B	Person[2].Job[1]
9	1	17	B	Person[2].Job[2]
10	1	19	B	Person[2].Job[3]
11	1	21	B	Person[2].Job[4]

Figure 12. IDs for Person and Job blocks (partial)

```

SELECT
  BP.JOINKEY,
  BP.INSTANCE AS PersNum,
  BP.NAME,
  BJ.INSTANCE AS JobNum,
  BJ.EMPNAME
FROM
  dbo.BLAISE_FORM AS BF
  INNER JOIN dbo.COMMUTE7MOD_BPERSON AS BP
  ON BF.ENDSTAMP = '9999-12-31 00:00:00.000'
  AND BF.JOINKEY = BP.JOINKEY
  AND BF.DMKEY = BP.DMKEY
  AND BF.BEGINSTAMP = BP.BEGINSTAMP
  INNER JOIN dbo.COMMUTE7MOD_BJOB AS BJ
  ON BP.JOINKEY = BJ.JOINKEY
  AND BP.DMKEY = BJ.DMKEY
  AND BP.BEGINSTAMP = BJ.BEGINSTAMP
  AND BJ.BLOCKID BETWEEN BP.BLOCKID + 1 AND BP.BLOCKID + 9
ORDER BY
  BP.JOINKEY, BP.INSTANCE, BJ.INSTANCE

```

Here again, automated views can be generated to hide the complexities of the SQL.

3.4.4 Beware of the Secret Data

At one point when we first started working with Datalink, we used SQL rather than a Blaise module to change some data values. After doing so, we were greatly surprised to look at the data with Blaise and still see the old values. The Statistics Netherlands team informed us that (in the words of the current documentation), “Fast reading is enabled by default for BOI files which support record streams. When using a BOI file that supports record streams then the complete data of a Blaise record will be stored as a binary stream in the relational database. ... if fast reading is enabled, then Blaise will read the record data from the record streams, instead of the data that is stored according [to] the given data partition type. This means that whenever you manually change the data in the 'normal' data tables, that you won't see the changes in the Blaise client tools like the DEP and Dataviewer.”

The documentation indicates that you can avoid this problem by disabling Fast Reading. We decided that the better part of valor was to limit our non-Blaise access to readonly queries, leaving all data changes in the hands of Blaise.

3.5 Tools

While we were developing the latest version of the Blaise Editing System, we found that deploying a new instance of the system still required a number of manual processes. In order to expedite the system deployment process, we devised a Generation Tool to automate the most repetitive parts of the process. This tool, as depicted in Figure 13, allows us to generate:

- the database schemas,
- the triggers for each table,
- the .BOI files and the Blaise generic tables,
- meta tables, and
- database views.

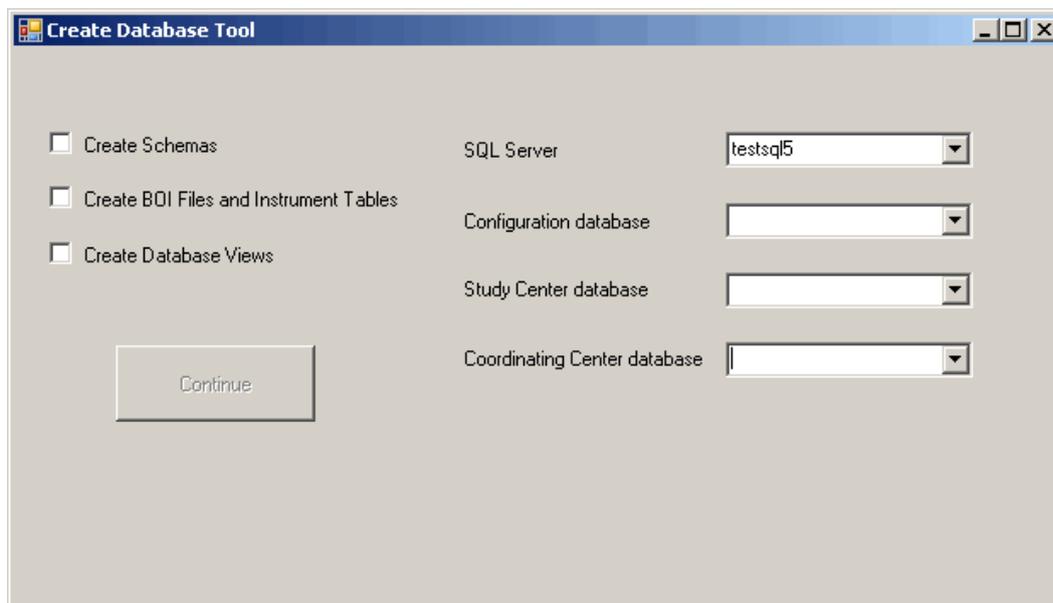


Figure 13. Generation Tool

The driver behind the Generation Tool is an .ini file. This file contains information about the SQL database, datamodels, schemas, the .boi template, and other study-specific items.

In addition to developing our own tools, we also found the new Blaise Data Center to be extremely useful for providing quick and easy *ad hoc* access to the data without having to write SQL queries or other programs (Figure 14).

	ASSIGNMENTID	ENBEGINTIME	ENCORRECTADDRESSSTRNO	ENCORRECTADDRESSSTR
▶	A1000XX-5	6:27 PM	1650	MAIN STREET
	A1000XX-5	6:27 PM	1650	123 MAIN STREET
	A1004XX-5	6:16 PM	1650	Street1
	A1005XX-3	3:23 PM	1650	Street1
	A1007XX-3	1:20 PM	1650	Street1
	A1009XX-3	1:30 PM	1650	Street1
	A100XX-4	7:36 PM	1650	Street1
	A1012XX-3	3:43 PM	1650	Street1
	A1013XX-3	11:41 AM	1650	Street1
	A1014XX-3	4:23 PM	1650	Street1
	A1017XX-1	9:20 PM	1650	Street1
	A1019XX-1	4:08 PM	1650	Street1
	A1020XX-9	7:36 PM	1650	Street1

Figure 14. Data Center Record View

One of the Data Center's most valuable features is its ability to export subsets of records and variables. We have also used it to verify the data changes made in the Blaise Editing System, as shown in Figure 15.

	Row	Item	Value #1	Value #2
	000003	BeginStamp	20090428 17:39:14.813	20090428 17:21:48.790
	000004	EndStamp	99991231 00:00:00.000	20090428 17:39:14.813
	000005	Form Status	blfsClean	blfsNotChecked
▶	000046	EnCorrectAddressStr	123 MAIN STREET	MAIN STREET

Figure 15. Data Center version differences

4. Conclusion

The latest version of Blaise Datalink, with its generic storage and versioning capabilities, provides a solid foundation for systems where tracking data changes is a requirement and you also need to query the data outside of a pure Blaise environment. The incorporation of Datalink into the latest version of our Blaise Editing System has resulted in a system that offers enhanced flexibility for our projects while also being simpler and easier to maintain.

5. References

Dulaney, Rick, and Boris Allan. "A Blaise Editing System at Westat." *Proceedings of the 7th International Blaise Users Conference*. September 2001.

< http://www.blaiseusers.org/2001/papers/Dulaney--Blaise_Editing_System_at_Westat.pdf > (April 26, 2009).

Gowen, Linda, and Pat Clark. "Lifecycle Processes to Insure Quality of Blaise Interview Data." *Proceedings of the 11th International Blaise Users Conference*. September 2007.

< <http://www.blaiseusers.org/2007/papers/D4%20-%20Lifecycle%20Processes%20to%20Insure%20Quality%20of%20Data.pdf> > (April 26, 2009).

Purohit, Yogeeta, and Latha Srinivasamohan. "Conversion of Blaise Databases to Relational Databases." *Proceedings of the 11th International Blaise Users Conference*. September 2007.

< <http://www.blaiseusers.org/2007/papers/P1%20-%20Conversion%20of%20Blaise%20Databases%20To%20Relational%20Db.pdf> > (April 26, 2009).

Rouschen, Arno. "Generic Data Storage with Blaise 4.8 Datalink." *Proceedings of the 11th International Blaise Users Conference*. September 2007.

< <http://www.blaiseusers.org/2007/papers/A2%20-%20Generic%20Data%20Storage.pdf> > (April 26, 2009).

Michigan Questionnaire Documentation System, Version 3 (MQDS-V3)

Karl Dinkelman, Nicole Kirgis, Gina-Qian Cheung, University of Michigan

1. Introduction

The Michigan Questionnaire Documentation System (MQDS) was designed to extract comprehensive metadata from Blaise survey instruments and render it as an eXtended Markup Language (XML) document using the Data Documentation Initiative (DDI) standard. The DDI standard is an international effort establishing metadata markup standards for expressing social science data. In April 2008, the DDI Alliance released the third version of specifications, which added rich support for documenting Computer Assisted Interviewing (CAI) instruments. The current version of MQDS has been updated to exploit many newly introduced features by changing the method of processing Blaise instruments in to a relational database. The process of generating documentation from a Blaise datamodel in MQDS V3 will be explained in this paper.

2. Background

Several papers have detailed the involvement of the Survey Research Operations (SRO), a unit within the University of Michigan's Institute for Social Research's Survey Research Center, in the development and creation of the MQDS application (Sparks and Liu 2004, Guyer and Cheung 2007). Sparks and Liu (2004) describe early development of MQDS and what was referred to as "BlaiseDoc" in the early stages of developments (or MQDS Version 1). Guyer and Cheung (2007) further explained development activities that lead to MQDS Version 2 and 2.5, rewriting the program in .NET, and described additional utilities. Ironically, in most cases, the new utilities were added to alleviate limitations experienced by users. For example, users had difficulty processing large-complex instruments, thus a utility named *XML Merge* was added allowing users to break processing into two or more steps by selecting sections or blocks within an instrument. The *XML Merge* utility then concatenated the multiple XML files into one XML file theoretically representing the output that MQDS would have been generated if it could have processed large-complex instruments.

The creation of the most recent version of MQDS, Version 3, has been fueled by several development initiatives as well as the limitations observed by users of previous versions of MQDS. The remainder of this section will focus on outlining three fundamental aspects that assist in setting the stage for the eventual creation of MQDS V3. These three areas include the fundamental limitations of previous MQDS versions, the newly released Data Documentation Initiative (DDI) version 3 metadata standard, and a collaborative project between the Interuniversity Consortium for Political and Social Research (ICPSR), a unit within the University of Michigan's Institute for Social Research, and SRO to create a database to store, share, and disseminate DDI metadata. ICPSR is the world's largest archive of digital social science data (<http://www.icpsr.umich.edu/>).

2.1 Limitations of Previous MQDS Versions

Although MQDS V2 offered many new, previously unavailable data documentation tools, the fundamental limitations found in previous version of MQDS were related to limited usability. The main limitations users experienced with MQDS V2 included:

- Many users with large-complex instruments found processing difficult (if not impossible) due to memory limitations. This was due to MQDS storing and processing redimensioned arrays in memory causing performance degradation over time as the memory usage continued to grow. Some found that to process large and complex instruments one would have to experiment by selecting fewer options (i.e. without goto's, without fills, etc) or breaking instruments into smaller manageable pieces (as described earlier with the XML Merge utility). However, this experimentation did not always result in positive outcomes, and for those users, MQDS remained unusable.
- Another drawback for the user was that they were required to select all the items they wanted to see in their output at runtime. If the user wanted to add additional items or remove some of them, they had to return and rerun MQDS to produce the desired set of output files.
- A small set of external clients, found it difficult to implement supporting tools around the core MQDS application. For example, tools that would allow one to repurpose the MQDS output or to polish it in the preparation of producing instrument documentation that would eventually be published in an online archive.

- The final output did not validate to the DDI V2 DTD. Document Type Definition (DTD) defines the set of elements (XML tags) and attributes (attached to a given tag) permitted in a given XML document. Wikipedia states that DTD's, "...explains precisely which elements and entities may appear where in the document and what the elements' contents and attributes are." (2009). Even though the MQDS V2 output contained 80 – 90 % of what would be considered a valid DDI V2 file; it remained noncompliant as many items needed by MQDS had not yet been added to the DDI specification and were not declared in its DTD.

2.2 The introduction of the DDI V3 Standard

As mentioned earlier, The DDI Version 2 standard lacked essential elements needed to accurately describe the Computer Assisted Interviewing (CAI) instrumentation process. The scope of previous versions of the DDI standard had traditionally focused on the archival process of data documentation and the creation of codebooks (Thomas et. al., 2008). However, the DDI Alliance's release of its third version of the DDI metadata standard in April 2008 modestly attempts to alleviate these issues. In fact, DDI V3 represents a dramatic shift towards documenting the overall survey life cycle; allowing for "a more coherent and richer description of the questions, their survey instruments, and the means of data collection" (Thomas et. al., 2008). Substantial work went into extending DDI V3 to encompass expanded elements describing questionnaires and the CAI instrumentation process (Dinkelmann, 2006). This work was the amalgamation of a work group consisting of individuals from more than 10 organizations who met via conference calls for over two years creating the proposed additions to the DDI V3 standard.

The extensions to DDI V3 come at a cost. Essentially, unless endless resources are at hand, the notion of marking up a DDI XML document manually has become extremely difficult. Therefore, for an organization to implement the DDI V3 into their processes it has become essential to utilize a mechanism for automating the creation of a DDI instance. A DDI instance could be thought of as a single, self contained XML document that validates to the DDI V3 XSD. Some extensions to the latest version include, but are not limited to, an extended set of metadata elements allowing the capturing of survey instrumentation logic (cf. the *ControlConstructScheme* element found in the Data Collection module in DDI V3). The Control Construct consists of a series of elements that are used to describe the sequence and flow of questions and supporting information within a data collection instrument (DDI 3.0 XML Schema Documentation, 2008). These elements include the following: *IfThenElse*, *RepeatUntil*, *RepeatWhile*, *Loop*, *Sequence*, *ComputationItem*, *StatementItem*, and *QuestionConstruct*. This new structure allows individuals who wish to document the survey instrument new opportunities. For example, a procedure is available in MQDS that reads the Blaise meta information file (or .bmi) to create Goto's for representing skip patterns in a given instrument. Previous DDI versions did not accommodate for documenting Goto's. However, the current DDI version allows for the storage of Goto information in the *StatementItem* element and references the *StatementItem* in the *IfThenElse* element.

DDI V3 moves from using DTD to XML schema language (XSD), often considered the successor of the DTD. Wikipedia describes XSD (2009) as follows:

"...can be used to express a set of rules to which an XML document must conform in order to be considered 'valid' according to that schema. However, unlike most other schema languages, XSD was also designed with the intent that determination of a document's validity would produce a collection of information adhering to specific data types. Such a post-validation *infoset* can be useful in the development of XML document processing software..."

Therefore, outside of the rules for validating a DDI V3 XML instance, it also moves us a step closer to ensuring interoperability between XML instances. Wikipedia goes on to state the following about the "namespace" feature offered in the XSD standard,

"The most obvious features offered in XSD that are not available in XML's native Document Type Definitions (DTDs) are namespace awareness, and datatypes, that is, the ability to define element and attribute content as containing values such as integers and dates rather than arbitrary text."

DDI V3 uses XML namespaces for the first time allowing users to modularize its content, offering an architecture where users can select modules within the scope of their given documentation projects (Thomas et. al., 2008). Furthermore, users can capture elements when and where they happen in the survey life cycle, aiding in the ability to pass metadata forward in the life cycle to those not directly involved in creating survey instrument and the data collection process thereby offering an unidentifiable number of ways to of reusing and repurposing of the metadata. To understand more about XML namespaces, we turn again to Wikipedia (2009):

“XML namespaces are used for providing uniquely named elements and attributes in an XML instance. They are defined by a W3C recommendation called Namespaces in XML. An XML instance may contain element or attribute names from more than one XML vocabulary. If each vocabulary is given a namespace then the ambiguity between identically named elements or attributes can be resolved.”

Other significant advances in the DDI V3 standard that MQDS will take advantage of include the explicit use of the XHTML standard and documenting content once and referencing it elsewhere. Additionally, others that might be incorporated into MQDS in the future include the ability to document version changes within survey instruments, extending the DDI V3 schemas by adding user-defined or MQDS namespace (for elements and attributes outside the scope of DDI), and a mechanism for grouping and comparing related survey instruments to document similarities and differences that exists.

2.3 Data Documentation and Dissemination Project

Beginning in late 2007, members of the MQDS development team collaborated with a team from the ICPSR to begin work on the data dissemination and documentation project. The goal of this project was to create a common database design for storing survey metadata that was based on the newly released DDI V3 standard. The common database would allow for the transfer of survey data and metadata between SRO and ICPSR seamlessly. An initial SRO/ICPSR joint vision of this project is shown below in figure 1.

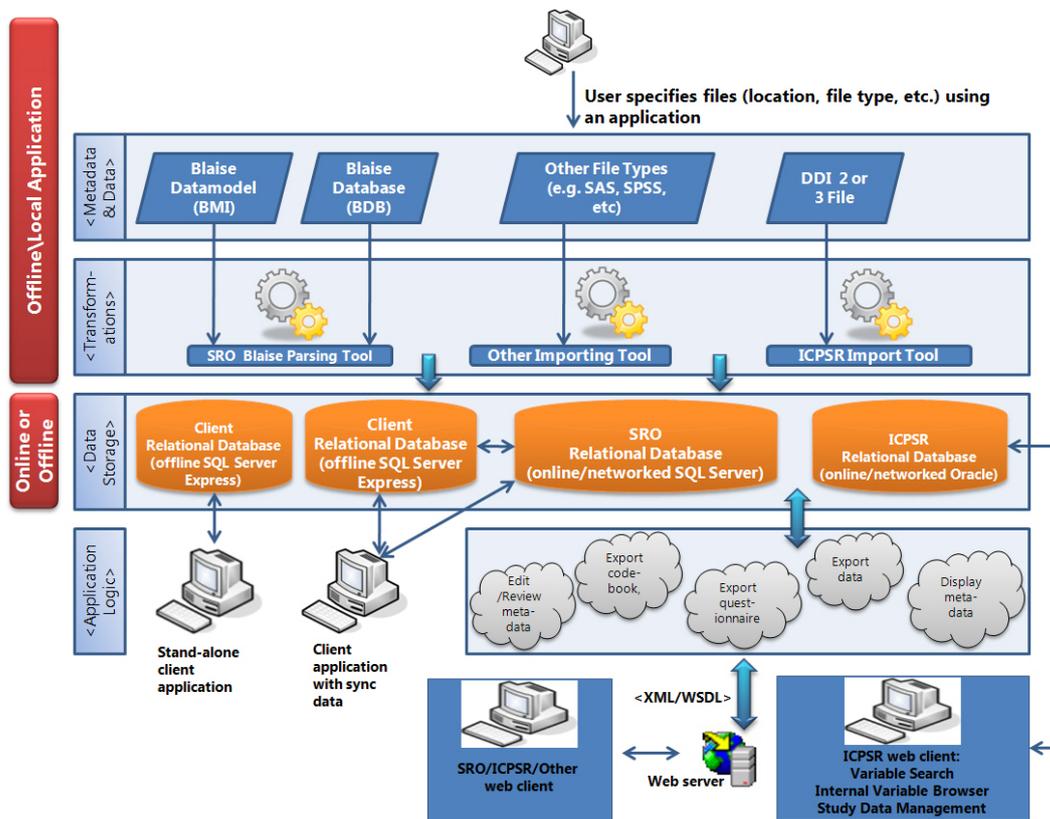


Figure 1: Initial SRO/ICPSR Joint Vision

The main outcome of this project was a database datamodel specification that has become the foundation of the new MQDS V3 database.

3. MQDS V3

The guiding principles in creating MQDS V3 were as follows:

- Address limitations identified in the previous versions of MQDS;
- Exploit new and expanded elements of the DDI V3 standard and to validate to its XSD;
- Move from the “in memory” processing model to streaming to a relational database model;
- Allow users to extend the MQDS datamodel to meet the needs of their project teams;
- Provide a tool that is closely aligned with the data collection and archival community’s needs;
- The ability to process Blaise instruments of any size.

The next several sections will discuss MQDS V3 as we envision it as the development efforts for V3 had not finished as of writing this paper.

3.1 Import-Export-Transform Process Model

One of the primary differences in MQDS V3 is the use of a database design, modeled after DDI V3, to ensure alignment between MQDS and the latest DDI standard. The basis of the MQDS V3 design can be explained using the Import-Export-Transform (IET) process models. This process model divides MQDS into three different core components. These components are illustrated below in Figure 2.



Figure 2: MQDS V3 – IET Process Model

The import component takes the Blaise BMI (and associated data and metadata elements) and loads them into the MQDS V3 relational database. The export component extracts the database content to a DDI V3 compliant XML file. The transform component then renders the XML file in an output format of the user’s choice (for example html representation of a questionnaire). The next three sections will describe these components.

3.1.1 The Import Process

The import process reads instrument data (summary statistics and frequencies) and metadata (*.bmi and *profile* data; profiles are explained in the next paragraph) into the MQDS V3 database. In redesigning the user interface for V3, the method of creating documentation was streamlined. Previous MQDS versions offered three different paths based on the type of output the user desired (i.e. questionnaire documentation, codebook from bdb, or codebook from SAS). The current version combines these into one operation.

The configuration file (used in previous versions) is replaced with *Profiles*, permitting the user to capture information and save it for future use, much like the previous configuration. However, the content captured and stored in the profiles is specific to items introduced in the DDI V3 standard. Two types of profiles have been added: the *Instance Profile* stores information about the current study and import session and the *User Profile* captures user information less likely to change overtime. The *User Profile* also store information about the database connection settings. The many of the items captured in the profiles have a direct mapping to DDI V3 elements. The use of profiles has been added to allow the user to overwrite some of the information that would typically be pulled directly from the *.bmi to allow for an XML instance that is more self contained.

Additionally, some of the items captured in the profiles are mandatory for the user to enter before they continue, the mandatory elements are set forth from elements that are deemed as mandatory within the DDI V3 schema definitions and are not always captured in a given *.bmi file. Figure 3 provides an overview of the import process into the MQDS relational database.

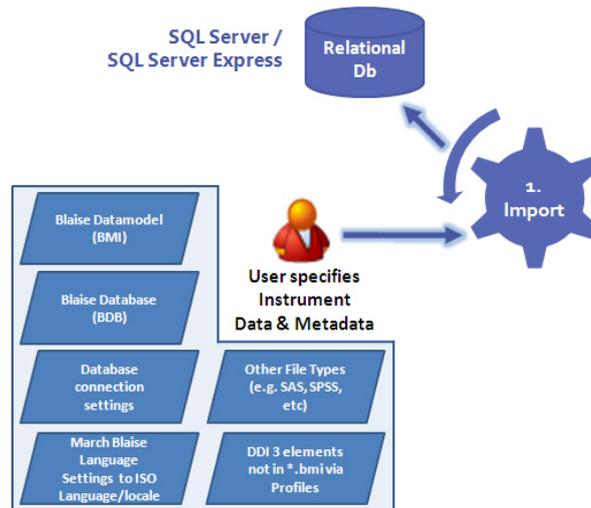


Figure 3: Import Instrument Data & Metadata

The MQDS V3 database design offers the ability to utilize database methods for queries, comparisons, and merging of data. MQDS V3 writes to either the freely available Microsoft SQL Server Express (2005 and 2008), typically residing locally on the user’s computer, or to a remote SQL Server instance residing on the user’s network. The remote SQL Server instance is a licensed version of Microsoft SQL Server. Nevertheless, we have focused the majority of our development on the SQL Server Express version.

Once the user has imported the instrument data and metadata into the database, the next step is to export data to DDI V3 XML instance discussed in the next section. However, this is also a point where the user could extend MQDS to meet their needs (described in section 3.3 Extending MQDS).

3.1.2 The Export Process

The export process creates a ‘valid’ DDI V3 XML Instance. During the export process, the user is asked to select the languages to export and to match ISO language pairs (i.e. Language-locale) that accurately reflect the language settings found in the *.bmi. The user can also select to export only the question fill names or the fill representation based on the Blaise rules and if the exported text should be formatted (based on the Blaise syntax presentation) or unformatted. They can elect to export formatted or unformatted text. Figure 4 presents an overview of the Export process for creating a DDI V3 XML Instance.

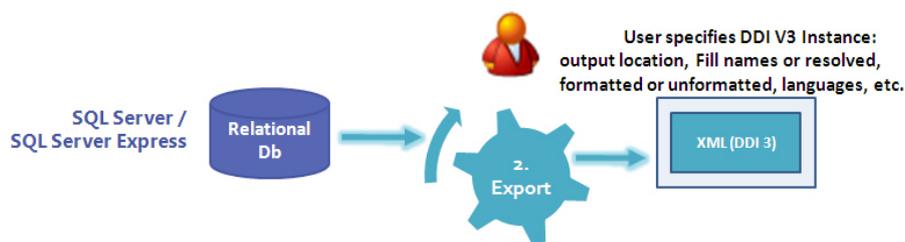


Figure 4: Export DDI V3 XML Instance

3.1.3 The Transform Process

The transform process begins by taking the DDI V3 XML Instance generated in the export process and allows the user to create understandable output. Figure 5 (shown below) offers a visual representation of the transformation process from a DDI V3 XML instance process. While working with this process, the user walks through a series of steps identifying the following options: the output location of the transformed content, the transformation format (HTML, RTF, or PDF), default stylesheet settings or a modified stylesheet. If the user selected to modify the stylesheet, a new window is presented allowing the user to select the elements to include

in their output (similar to the stylesheet section window in previous versions of MQDS). A new output feature has been added to MQDS at the request of users offering the option of creating a question/variable index listed with appropriate links to its output location facilitating faster searches of the output content. This is listed at the beginning of the output, on a separate page and presents the questions by blocks or sections thus facilitating faster searching of the output content.

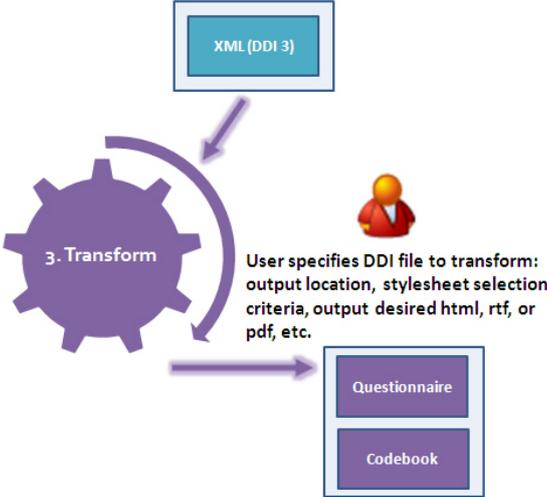


Figure 5: Transform DDI V3 XML Instance

3.2 The IET Process Model & MQDS

Figure 6 shown below illustrates how the three parts of the MQDS V3 IET process model work together.

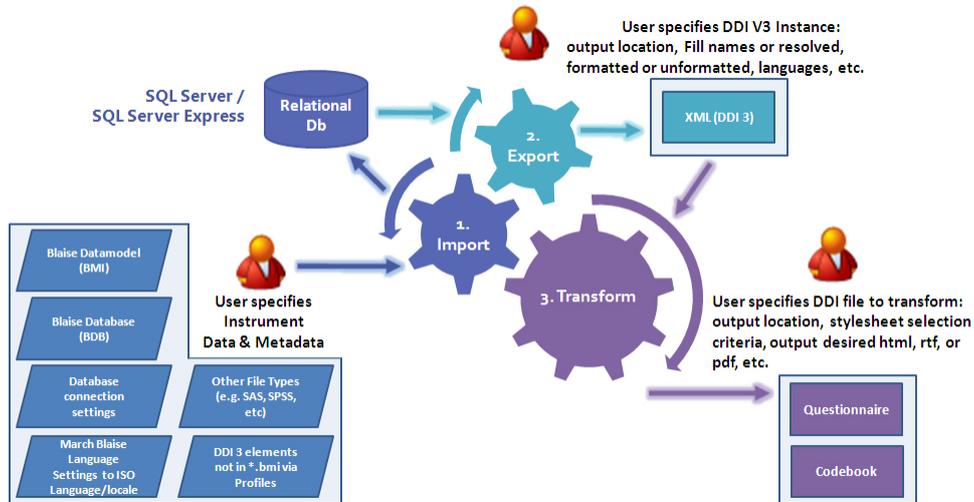


Figure 6: The MQDS IET Process Model

Despite the simple nature of the development of MQDS V3 and its use of the DDI V3 standard, development was quite complex. The initial development plan called for reusing much of the code used in Version 2 whereas full sections were rewritten from the ground-up. MQDS V3 development began before the final version of the DDI V3 standard was released. Much of the xml that MQDS needed to create had never been created before. As there were no examples to follow, new examples were manually created.

3.3 Extending MQDS

While designing MQDS V3 and attempting to address the limitations experienced in previous versions of MQDS, the development team attempted to generate use cases for the possible ways users would want to extend the application. The image below in figure 7 illustrates only a fraction of the notion of extending MQDS, as its new database structure offers the user with many other opportunities that will not be discussed here. The arrows pointing up in the image are intended to demonstrate possible points where users could extend MQDS by creating supplementary tools.



Figure 7: Extending MQDS

For example, the user could extend the import component by creating a database-editing tool allows direct access to the MQDS output allowing for direct modification of the metadata. Possible areas where the user might want to modify the output include modifying field descriptions (or SAS labels) for a more readable format for data out

purposes or adding human-readable universe descriptions (as MQDS currently exports machine-readable universes descriptions).

4. Known limitations of MQDS V3

Despite efforts to combat the known limitations in previous versions MQDS, some not so obvious limitations still exist within the output generated from MQDS V3. These limitations can be classified into two separate categories, one for documenting instruments programmed in Blaise and those imposed by DDI V3. MQDS V3's limitations in documenting instruments programmed in Blaise center on the expanding uses of the Blaise language setting. MQDS V3 currently makes the assumption that the use of languages within Blaise applications is used for the purpose of expressing different languages (i.e. English, Dutch, Spanish, etc). The *Blaise 4.8 Online Assistant* defines "Languages (setting)" as, "...the form languages available in a multilingual data model." (2009). However, the use of the language feature in Blaise currently extends past the semantic interpretation of language. For example, a user can use the language feature to define different modes in a mixed-mode survey, add help and metadata content, multimedia content, or different instrument versions within the same study. Unfortunately, unless the Blaise software were to force language identifiers, such as HLP for help and MDL for metadata language, it becomes difficult to determine that these are not real languages. The only way to tell currently is by looking to see if the given language identified in the data model is set as a spoken and unspoken language. This means that the methods for assessing how the language function is being used within a Blaise application are limited. Therefore, they are processed as though they are actual languages until a more elegant solution is devised. The interesting aspect here is that MQDS is able to create DDI V3 instance representations that are syntactically valid yet they remain semantically incorrect. It is important to note the paradoxical nature of this limitation, that is, even though it poses a problem in creating systems that support instrument documentation, we could develop equal counterpoints to explain how this could be considered one of the systems most valuable features. Nevertheless, we must attempt to devise a solution that suits the majority of users.

Most of the limitations brought forth as a result of integrating the DDI V3 standard are still unknown, as the MQDS's implementation of its most recent version has only scratched the surface. There are several known issues related to Blaise. For example, Blaise does not currently offer an explicate way of documenting *Alien Procedures* (calls to external routines) and *Alien Routers* (calls to external programs). Additionally, DDI currently does not offer ways of documenting paradata associated with the instrument. However, the DDI standard is an on-going process with mechanisms in place for submitting proposals for additions to the current standards. Additionally, the ability to extend the schema by adding additional namespaces exists. For example, we could add a MQDS or Blaise namespace to accomplish documenting these areas.

5. Conclusion

Despite the lengthy development time, many of the limitations experienced with the previous version of MQDS are now resolved leading to improved usability. Although limitations will be identified with MQDS V3, the many gains of the latest version should outweigh the limitations.

6. Acknowledgements

The authors would like to thank the following individuals from SRO for their contributions towards the development and creation of MQDS V3: Sheila Deskins, Nathaniel Doran, David Padot, Ruth Philippou, Peter Sparks, and Isabella Yeh. Thank you to the ICPSR team for their collaborative efforts in creating the shared datamodel (a.k.a. the "data dissemination and documentation project").

7. References

Blaise 4.8 Online Assistant (Version 4.8.1.1339). (2009).

Data Documentation Initiative (DDI). <http://www.ddialliance.org/>, Retrieved Apr. 28, 2009

DDI 3.0 XML Schema Documentation (2008-04-28). (2008) Windows Help file.
http://www.icpsr.umich.edu/DDI/ddi3/DDI_3_0_Documentation_WindowsHelp.zip, Retrieved Apr. 28, 2009

Dinkelmann, K. (2006). DDI Version 3 and Instrument Documentation. *The proceedings of the 32nd International Association for Social Science Information Services & Technology (IASSIST)*. Ann Arbor, Michigan.

Document Type Definition. (2009, May 7). In *Wikipedia, the free encyclopedia*. Retrieved April 28, 2009, from http://en.wikipedia.org/wiki/Document_Type_Definition

Guyer, H. & Cheung, G. (2007). Michigan Questionnaire Documentation System (MQDS): A User's Perspective. *The proceedings of the 11th International Blaise Users Conference (IBUC)*. Annapolis, Maryland.

Microsoft SQL Server 2005 Express Edition. <http://www.microsoft.com/Sqlserver/2005/en/us/express.aspx>.

Microsoft SQL Server 2008 Express. <http://www.microsoft.com/express/sql/>

Microsoft SQL Server 2008. <http://www.microsoft.com/sqlserver/2008/en/us/default.aspx>

Sparks, P. & Liu, Y. (2004). Blaise Documentation System. *The proceedings of the 9th International Blaise Users Conference (IBUC)*. Gatineau, Québec.

Thomas, W., Gregory, A., Gager, J., Kuo, I., Wackerow, A., and Nelson, C. (2008) Data Documentation Initiative (DDI) Technical Specification Part I: Overview (Version 3.0) (document version 7). Copyright DDI Alliance 2008.

XML Namespace. (2009). In *Wikipedia, the free encyclopedia*. Retrieved April 28, 2009, from http://en.wikipedia.org/wiki/XML_Namespace

XML Schema (W3C). (2009). In *Wikipedia, the free encyclopedia*. Retrieved April 28, 2009, from http://en.wikipedia.org/wiki/XML_Schema_Definition

Tracking And Rotation – The Next Steps

Jo Edwards, Office for National Statistics

1. Introduction

As stated in Setchfield, 2007, in order to meet new requirements for the European Union's Survey of Income and Living Conditions (EU-SILC), the General Household Survey became longitudinal in 2005. It is now called the General Lifestyle Survey (GLF). The result was the implementation of a rotating four-year panel design. Whilst longitudinal social surveys were not new for ONS, this was the first one that followed all adult persons with the original sampled household for four years, unless they died, moved to an institution or moved abroad. Other characteristics include interviewing adult household members if they join after wave 1 as long as they live with an original sampled household member.

In 2006, a new method of rotating data, and tracking household members was developed. This method catered to the addition and removal of household members, including moving non-residents into new households for either interviewing in the case of movers or processing in the case of those who became ineligible.

Since this time, ONS has taken on two new longitudinal surveys – the Household Assets Survey (HAS) and the Lifestyle Opportunity Survey (LOS). Like GLF, both of these surveys follow all adult members of the original sampled household. HAS follows up eligible respondents after 2 years. However, if respondents show certain characteristics, they are followed up after 1 year for a Household Debtors Survey, which is conducted by telephone. LOS follows households after 1 year.

A Keeping-in-Touch Exercise (KITE) survey is run about 3-4 months prior to the face-to-face survey for all the surveys. The KITE survey checks for movers, update address and contact information, and in the case of HAS checks for joiners and collects basic information, and in the case of LOS will additionally check for onset of disabilities. In the case of LOS, if onset of disability is recorded, that household becomes eligible for a face-to-face interview at the next wave.

ONS is reviewing current processes and practices in relation to its longitudinal social surveys. Several gaps and improvements have been identified, and work is ongoing to define the issues more thoroughly so that appropriate solutions can be implemented. This paper will primarily concentrate on the GLF and will discuss some of these gaps and solutions, although at the time of writing, they are yet to be finalised and implemented.

2. Current Methods.

GLF and HAS both currently use the same methods for tracking and rotation. The method was detailed in Setchfield, 2007. The four main areas this paper will consider are: use of an external database to hold previous waves responses; filter block; concertinaing of remaining household members; and the Keeping-In-Touch-Exercise (KITE) survey. Current practices for these methods are summarised below.

2.1 Use of external database

The method of having the previous wave's survey data in an external file was chosen over rotating directly into the fields. This allowed several other options to be considered. Firstly, to allow consideration of having an option to choose people from different addresses, that is, to allow for merged households. Secondly, rotation could be processed within the datamodel. And lastly, in most cases the information was fed forward into question text or used as checks rather than pre-filling answers.

Subsequent analysis suggested merged households would be rare, so the option of allowing individuals to be picked from different addresses was dropped.

The external file holds information at both the household and person level.

A Keeping-in-Touch Exercise (KITE) survey is run about 3-4 months prior to the face-to-face survey. See section 2.4 for more information about the KITE survey. To alleviate the need to merge the KITE database with the previous wave's database, the previous wave's database is rotated into KITE. Thus the resulting KITE database is turned into the external file for the next wave. The data is rotated into a rotate block, which also executes some rules that the survey has on person level rotation. Then questions that use rotated data refer to this block.

2.2 The Filter Block and Splitting Households in the Field

A filter block was developed to ascertain the eligibility of all household members who were present in the previous wave, prior to the interview taking place. For those members who are no longer present, extra information such as date they moved, and contact details if they were still eligible (that is, moved within UK to another private household) or date of death are collected.

A summary screen then alerts the interviewer to whom will be interviewed in the current household, as well as alerting them to the extra households they need to create for the movers/ineligibles. The filter block is repeated on the newly created households. Currently the interviewer has to fill in the filter block on the additional households. We are looking to see if it can be automatically pre-filled correctly. That is, if the newly created household is eligible for interview, then the residents in the newly created household will have a status of resident – whereas in the original household they will have the status of mover. An interviewer cannot transmit back a case if they haven't created the required number of households.

The method of opening up extra households for those no longer resident/eligible, was used to process people from the last interview into similar categories. The categories were those who have moved locally, which the interviewer would go and interview, those who moved within UK, which would get reallocated out to an interviewer in that area, and those no longer eligible, such as those who moved abroad, moved to an institution or died.

2.3 Concertinaing the Household for Interview

Once those no longer resident in the household are removed, all remaining members are put together for the interview. Within the rotate block, there is code that matches the interviewee position number with their position number in the Filter grid. This ensures that the correct information is rotated forward for the interviewee. It also assists the interviewer visually. By grouping together those who are to be interviewed, no spaces are left in the answer pane. This is particularly useful for large share houses where there can be a lot of movement in and out of the household.

2.4 KITE

About three months prior to the main interview, and as part of the Keeping in Touch Exercise (KITE), a telephone survey is conducted to check on respondents. The main purpose is to check for movers, so that interviewers can go to the correct location at the main survey. Due to the current telephone unit systems, households are not split during the interview if household members move or become ineligible at the KITE survey. Instead, movers are flagged and put into new households when the sample is created for the next main survey. Currently ineligibles are ignored at KITE and remain in their current household. In the case of HAS, the KITE survey also gets information about joiners, as well as some basic demographic information. In the case of LOS, the KITE survey will also check for onset disability in households where members don't have any disabilities to check whether an interview is required at the next wave.

In terms of tracking respondents, the KITE survey is an integral part of the process. However, current processes mean that KITE facilitates sample loss. As new households are created for movers, this means they get a new serial number. However, this is done manually in the sample system and not in the Blaise questionnaire or the Blaise database. To implement the new ID calculations, we need to address the mover process at KITE. In addition we are restricted in developing new systems unless they are within the Blaise environment.

3. Gaps

As part of the development of LOS, current methods and processes for longitudinal social surveys were informally reviewed. LOS were looking at both GLF and HAS to see what processes they needed to set up or develop. As part of this, several issues came to light.

For example, in order to prepare data for EU-SILC, several manual processes were introduced on the GLF. For instance, recreating the original household to show movers, matching households and people between waves if they were movers, and removing households, which were created due to their ineligibility at the filter screen are mostly done manually. Not only are these manual processes very time consuming, but they also allow errors to creep in.

The main issues, together with suggested solutions are described in the following sections.

3.1 Tracking for Weighting

Every year on GLF, a yearly file is prepared and sent to Methodology so that they can produce weights. In order to do this work, there are several pieces of information that they require.

Firstly, information on the status at the current wave of everyone who was in the household in the previous wave is required. This enables attrition and movements to be looked at.

Secondly, the members who have moved should be recorded both in the household where they were resident at the previous wave and the household where they are now resident. That is, they should appear in two households.

Thirdly, all persons from the previous wave must be recorded whether they are contacted/respond or not. This means that unallocated quotas, deaths, and moves to institutions/abroad need to be included in the file.

Lastly, all persons who have been lost (i.e. we don't know whether they have moved to a private household in UK) or become ineligible (i.e. moved abroad, died, moved to institution) should remain in the same household, but be coded at a person level as lost or ineligible. That is, a new household should not be created for these persons, as this suggests they have moved to a private household in the UK then later became ineligible.

To implement these requirements we need to make some changes to our processes. Firstly we need to modify whom we create new households for. We only need to do this for movers to eligible households in the UK. That is, for those that die, move to an institution or move, but location unknown, we do not create a new household but leave them where they are.

For the actual interview, we have two options for leaving in these ineligible household members. We could follow our Labour Force Survey (LFS) method that leaves these people in the same position in the household grid, but just blanks them out for the purposes of interview. Instead of a name, they put a description in, like "Moved to Institution". This can get unwieldy, particularly in group households that can change dramatically year to year. It should be noted that for LFS, it is repeated every quarter for 5 quarters so generally doesn't experience as much change to household membership like the yearly (and in the case of HAS, two-yearly) surveys do. The other, and preferred option at this stage is to keep the interview grid as it is currently – concentrating the household members such that only eligible members appear for the interview. Some of the problems this creates can be overcome by better use of the IDs, as discussed in the next section.

Lastly, we have been looking at creating a tracking block that contains information on the membership of the household in all waves. We are currently concentrating on requirements for GLF, which has 4 waves. The block will be an array of 4, each representing a wave. The information captured is likely to be HHNO (see section 3.2 for details), Household ID, Split Number (see section 3.2 for more details), then an array of household members containing Name, Age, Sex, Status at that wave, date of death/move if appropriate, Person No, Person ID. Ideally if they have moved to a new household, which is still eligible, it would be great to have the household ID of their new household.

3.2 ID's

A big problem area is identification of members and households. Currently we use a serial number made up of an area code, an address number and a household number as the identifier for households. For wave 1 the household number is always '01'. The area code is created during the sample creation and relates to a physical location within the UK. The address number is a sequential number from 1 to the maximum number of cases an interviewer will carry out in a particular month. Due to the large sample size of HAS, a particular area may be in the sample twice during a 2 year period. Hence within a wave, serial number is not always unique. However as it is unique within a month, we include the last two digits of year (e.g. '09' for 2009) and numeric month (e.g. '06' for June).

For example, if Area code = 05000, Address = 10, and household = 01, and the survey takes place in June 2009, the serial number will be 0906050001001.

In each subsequent main wave survey, new household can be created for movers in the field by the interviewer. When a new household is created, the household number is incremented by 1, and thus the serial number changes.

For example, if there is a mover from the household represented by Area code = 05000, Address = 10 and household = 01 in June 2009, then a new household is created which will have household = 02. The serial number becomes 0906050001002.

When the sample is created for the next interview – either KITE or main survey, the serial numbers are looked at. Due to the sample and allocation system, household number must equal 1 prior to the case being allocated and scattered to the interview. If a person moves to a different area, then they will receive a new area and address numbers, with household number set back to 1. If a person moves to a house within the same area, the area code stays the same but they get a new address number and household number is set back to 1.

Rather than keeping the serial number from wave 1, it is recreated at each interview (both KITE and main interviews). The sample file contains current serial number and “lastserial”, which we can use in matching. “Lastserial” refers to the serial number at the last interview. As we use serial number as a proxy for household ID, the recreation of serial number each time makes it difficult to match between waves, particularly for movers. Using Manipula, we currently match from a wave back to the KITE then back to the wave and so on. Then manually, any households that were dropped from the sample, either because they were refusals, or ineligible or movers with unknown contact details are added back in. Once this is done, we can use the resulting file as look-up across all waves so that wave 4 can be matched to responses in wave 1.

With the introduction of LOS, we have taken the opportunity to develop a different method, which will be used on all longitudinal surveys. Rather than using serial as a proxy for household ID, and recreating it each wave, we will create a new variable called HHNO in wave 1 that will remain with the household (and those that split off) through the life that they are in the survey. In addition we will implement the ‘split number’ concept as per EU-SILC guidelines to identify those who have split from the original household. The household ID will then be made up of HHNO and the split number. The split number for the original sampled household is ‘00’. Rather than using a random number start to create HHNO, we will still utilise the last two digits of year, the numeric month, the area code and address number in wave 1 HHNO. HHNO will stay with all original sample households whilst they are in the survey regardless of whether they move or not.

Example 1: household at Wave 1, Surveyed June 2007

Area = 05000			
Address = 10			
Household = 01			
Name	Date of Birth	Sex	Status
Bob	10/03/1963	Male	Resident
Mary	24/09/1965	Female	Resident
Tom	15/10/1987	Male	Resident

So in example 1 HHNO = 07060500010, and the split number = 00, as it is wave 1. Hence household ID = 0706050001000.

The second issue is the split number. This is proving more difficult to do automatically in the field. The main complication is the KITE survey and the fact we manually split out movers at this point. Initially in Wave 1 the split number is 00. Theoretically, every time a split occurs, the split number increases. As KITE is not done in the casebook environment, new households are not created at point of interview when movers are identified. Rather they are flagged and during the sample creation process new households are created for the movers. It is at this point that the split number will need to be updated for new households. Due to the current sample creation process, this will need to be done manually. For movers at the main wave surveys, it initially looked like we could create the split number automatically at point of interview when new households are created in the field. However, we quickly realised that if the original household had another mover and the already split out household had a mover, they would end up with the same split number. So again, this means households with movers will need to be identified and split numbers calculated once the cases have been returned to HQ. In the short term, the calculation of split numbers is likely to be done manually, although it is preferred that an automatic solution will be developed.

Example 2: household at Wave 2, Surveyed June 2008. Following on from example 1, Tom moves out. The household now looks like:

Area = 05000			
Address = 10			
Household = 01			
Name	Date of Birth	Sex	Status
Bob	10/03/1963	Male	Resident
Mary	24/09/1965	Female	Resident
Tom	15/10/1987	Male	Moved out

HHNO in the first household remains the same: 07060500010, and the split number is still 00. Household ID = 0706050001000

A new household is created for Tom.

Area = 05000			
Address = 10			
Household = 02			
Name	Date of Birth	Sex	Status
Tom	15/10/1987	Male	Resident

HHNO in the second household is the same as the first household: 07060500010. However the split number changes to 01. Household ID = 0706050001001.

Lastly there is the issue of person number and person ID. Person ID was made up of household ID and person number. Again by default, we have been using position number in the interview grid to act as a person number. This is unique in a wave, but can be used by several different people across waves. Surveys are mainly interested in Person ID rather than person number so this hasn't been much a problem. In addition due to fact serial number was recreated at each wave, personal IDs for those joining a household always had a different year and month that those who were already in the household, so personal IDs were never duplicated even though person number was. However, changing the way household ID is created and re-looking at EU-SILC rules, we realised that we were not calculating person number correctly. Person number should not be reused within a household. Also, movers take with them, their person number. Joiners to households get the highest, unused person number. For more information see the EU-SILC paper referenced at the end of this paper. Along with changing household ID, we will implement the creation of person number and person ID at the same time. Person ID will still be made up of household ID and person number. Person number, in wave 1, will be the position number in the interview. Then in subsequent waves, person number for joiners will be the highest unused person number.

Example 3: Using the household in example 1. Person number is equal to position number in the first wave.

HHNO = 07060500010				
Household ID = 0706050001000				
Name	Date of Birth	Sex	Status	Person No
Bob	10/03/1963	Male	Resident	1
Mary	24/09/1965	Female	Resident	2
Tom	15/10/1987	Male	Resident	3

So Person ID of Bob will be 070605000100001.

In example 2, Tom had split out into a new household. Let's say he is joined by Jane.

Tom keeps his person number from the first time he was interviewed, so it is 3. Jane takes the highest unused person number in the household, which is 1 in this case.

HHNO = 07060500010				
Household ID = 0706050001001				
Name	Date of Birth	Sex	Status	Person No.
Tom	15/10/1987	Male	Resident	3
Jane	05/04/1988	Female	Resident	1

Tom's person number was assigned to him in Wave 1. So his person ID is 07060500100003. Jane's person ID is 070605000100101.

3.3 Rotation – Next Evolution

As mentioned in Setchfield, 2007 and summarised in section 2, the current rotation method uses an external file, which contains all the cases from the same reference month in the previous wave. This method is used for both HAS and GLF. Due to data confidentiality concerns as well as size concerns, we have begun to look into other methods.

As part of the Integrated Household Survey project, Manipula was used to rotate cases and data into those cases in HQ prior to the interview. This meant that the cases would go out with their data only. The method was developed on the Labour Force survey, which has different rules for rotating data. However, we are confident that we can use the same approach on GLF, HAS and LOS.

The main difference is that the process for which households to rotate occurs in the sample creation process. That means we only need to write a Manipula script for which variables to rotate data for. The data will be rotated into a holding block. This means we can still utilise the same filter and rotate blocks as we currently do. The only difference is that instead of referencing an external file, the holding block will be referenced. It is hoped that this new method will be implemented on HAS within a few months, then GLF and will be implemented for wave 2 of LOS which occurs early 2010.

Initially, it will be implemented in the main face-to-face survey case creation process. Unfortunately, at this stage the case creation process between the face-to-face survey and the telephone unit survey and also between surveys are not necessarily the same, so we have had to prioritise.

4. Next Steps

In addition to the solutions mentioned above, consideration is also being given to creating a tracking database. From this database, it is envisioned sample files can be created for KITE surveys and main wave surveys, data files can be created which go to methodology for the creation of weights, and updates can be made to household information, such as new addresses which are notified to ONS via change of address cards or other keeping in touch exercises can be applied. All sampled households will be loaded to the database. This is still in the initial discussion phase and it is not clear how urgent the need will be after the solutions discussed above are implemented.

The issue of updating household information as it is notified to HQ outside of surveys – such as receipt of change of address cards, individual refusals via the public enquiry line, notifications of deaths – is one outstanding issue that needs to be addressed. These notifications are generally small in number. A short-term solution is required, and the current idea is to implement a block that only comes online when the case is opened in HQ. To do this, we would utilise the edit version of the questionnaire. This block would be automatically pre-filled with the household information as at the end of the last interview, but would allow HQ staff to update fields such as status or contact information prior to the next survey. It is hoped a trial version of the block will be made available to HAS in the next few months to test.

5. References

Setchfield, C. (2007): Coping with People who just won't stay put: The Use of Blaise in Longitudinal Panel Surveys. Presented at 11th International Blaise Users Conference, 2007, Annapolis, September 2007

Eurostat (2008): Description of Target Variables: Cross-sectional and Longitudinal.

Some Techniques In Blaise Data Editing

Rob Groeneveld, Statistics Netherlands

Showing the number of errors in a form in the DEP

When editing a data file, it can be desirable to show the number of errors in a form on the Data Editing screen. As an example, we have this Blaise block:

```
BLOCK Blk_Person
FIELDS
  Name      "What is your name?":      STRING[20]
  Gender    "Are you male or female?": (Male, Female)
  Age       "What is your age?":      0..120
  Children  "How many children do you have?": 0..25
RULES
  Name
  Gender
  Age
  CHECK
  Age < 21 "Do not interview older people"
  IF (Gender = Female) AND (Age > 15) THEN
    Children
  ENDIF
ENDBLOCK { Blk_Person }
```

And a Blaise database:

Name	Gender	Age	Children
Kevin	Male	33	1
Anne	Female	34	
Nick	Male	44	2

This database contains inconsistencies: the males should not have children (a route error) and the ages of all three persons are not below 21 years, so they all have the hard error caused by the Age being over 21. We want to report these errors in the form of error counters on the screen. It is also an option to store these errors in the database for later retrieval. The checks on the errors in the record will be done outside the block, so the rules are checked wholly inside the block. We introduce four fields in the main datamodel:

```
HardError "Number of hard errors": INTEGER[3]
SoftError "Number of soft errors": INTEGER[3]
RouteError "Number of route errors": INTEGER[3]
NumberOfErrors "Total number of errors": INTEGER[3]
```

We could be tempted to try this in the rules:

```
HardError := HardError + 1
```

To count the number of hard errors. However, this could result in an increase in the field HardError each time a field in the record is changed because the rules of the datamodel are executed every time.

It is better to use the function GETERRORINFO on the record being edited. This is a function which can be applied to a file identifier in Manipula. For example,

```
Temp1.GETERRORINFO(1, 'KIND')
```

produces a string denoting the kind of the first error in the current record in the file Temp1. It can be a HARD, SOFT, ROUTE, SUPPRESSED or an IMPUTATION error. In order to use this function we must resort to a Manipula (or Manipulus) alien procedure. This is declared in the fields section in the main datamodel:

```

PROCEDURE NErrors
  PARAMETERS
    EXPORT pHardError
    ALIEN('TestErrors.msu', 'NErrors')
ENDPROCEDURE

```

And is invoked in the rules section like this:

```

NErrors(HardError)
HardError.SHOW

```

To show the field HardError as calculated in the procedure. We need a definition of the procedure in a Manipula setup which we will call TestErrors.man. This procedure has at its start

```

PROCESS TestErrors
USES
  ErrorModel 'TestErrors'
TEMPORARYFILE
  Temp1: ErrorModel
SETTINGS
  INTERCHANGE = SHARED

```

And defines the procedure NErrors for the parameter HardError as follows:

```

PROCEDURE NErrors
  PARAMETERS
    EXPORT HardError: INTEGER
AUXFIELDS
  ErrorIndex, TotalError: INTEGER
INSTRUCTIONS
  TotalError := Temp1.ERRORCOUNT
  FOR ErrorIndex := 1 TO TotalError DO
    IF Temp1.GETERRORINFO(ErrorIndex, 'KIND') = 'HARD' THEN
      HardError := HardError + 1
    ENDIF
  ENDDO
ENDPROCEDURE

```

The other error counters are calculated in the Manipula procedure in the same way. After implementing all this, the data editing screen for Kevin looks like this:

Name	Kevin	
Gender	1	Male
Age	33	
Children	1	
HardError	1	
SoftError	0	
RouteError	1	
NumberOfErro	2	

The numbers of hard errors, soft errors and the total number of errors appear on the screen.

Cross-record data editing in the DEP

Fields in another record can provide additional checks on fields in the record being edited in the DEP. Using a Manipula procedure, editing is possible with links to other records in the same or a different file, either a Blaise file or a BOI file. This increases the range of data editing problems to which the DEP can be applied.

For example, companies were asked about sales and purchases of various goods at various quantities and prices. Their answers were not always consistent with one another: when the same transaction was involved, one company sometimes reported a different amount or price than another company. The data have been entered into a Blaise table and it is now up to the data editor to reconcile the various reports when they conflict with one another. The data editor can decide to trust one of the companies involved or use some other means to make the data consistent.

As an example of the power of Manipula procedures, we present here a table with data on buying and selling various drinks. Various people bought and sold drinks to one another and reported on them. However, the price the seller reports on the sale differs from the price the buyer reports on the purchase of the same commodity. The table contains on both prices and presents to the data editor the choice of adjusting the selling price to the purchase price or vice versa, or no to make no adjustment.

This is the table after the data entry phase:

Name	ArticleSold	SellingPrice	Buyer	ArticleBought	PurchasePrice	Seller
Anne	juice	0,90	Nick	milk	0,60	Sandra
Kevin				coffee	0,70	Laura
Laura	coffee	0,55	Kevin	tea	0,65	Nick
Nick	tea	0,80	Laura	juice	0,75	Anne
Sandra	milk	0,85	Anne			

The interpretation: for example, Anne sold juice to Nick at the price of 0,90, while Nick reported on the same sale a price of 0,75. Anne bought milk from Sandra at a price of 0,60, but Sandra reported selling milk to Anne at the price of 0,85. Now it is possible that Anne or Sandra reported the price correctly and the other one made a mistake. The data editor has to base her decision on some criterion, using outside knowledge, for instance, she knows that Anne reports more reliably than Nick, or a paper receipt is available on the sale of milk by Sandra which settles the matter.

The datamodel BuyAndSell.bla contains the definitions of the fields in the table above and a call to a Manipula procedure:

```
PROCEDURE AdjustOtherField
  PARAMETERS
    Dummy1: STRING
  ALIEN('BuyAndSell.msu', 'AdjustField')
ENDPROCEDURE
```

The dummy string parameter is necessary because a procedure must have at least one parameter.

In the RULES section this procedure is called:

```
AdjustOtherField('')
```

The Manipula setup BuyAndSell.man uses the same datamodel and the setting INTERCHANGE = SHARED:

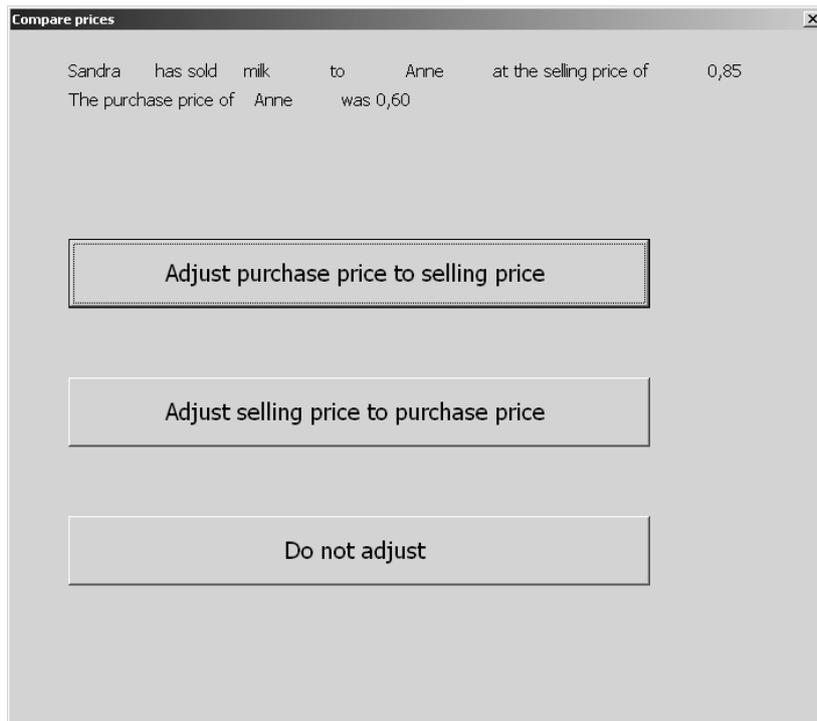
```
PROCESS BuyAndSell
USES
  BuyAndSellModel 'BuyAndSell'
UPDATEFILE
  Upd1: BuyAndSellModel('BuyAndSell', BLAISE)
```

SETTINGS

INTERCHANGE = SHARED

Note the use of the updatefile. The procedure AdjustField is also in this setup. It uses various other procedures and dialogs, among them the dialog Main, which asks the editor if she wants to compare prices or not, and if yes, reports the purchase and selling prices for the commodities a given person has sold or bought and presents three possible actions: to adjust the selling price to the purchase price, to adjust the selling price to the purchase price or not to adjust the prices.

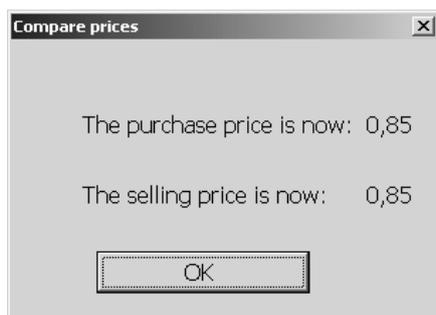
If the data editor wants to compare the prices the dialog Main is shown:



Behind the scenes, this involves a search and read of the record for the seller:

```
IF Upd1.SEARCH(Seller) THEN  
  Upd1.READ
```

The various prices are assigned to auxiliary fields and if the first button is pressed in the dialog Main the purchase price is made equal to the selling price and the records written back to the updatefile. The result is reported in another dialog:



The files of these examples are available from the author.

Table-like presentation in the Data Entry Program

A table presentation is a handy tool to group related data in a form. The syntax for a table in a datamodel is the same as for a block, except for the word TABLE at the start of the block (and ENDBLOCK or ENDTABLE at the end of the block). With some effort, the same effect can be obtained by special values in the Mode Library. As an example, we have a datamodel about turnover, profit and loss in companies. Each record contains the data for turnover, profit and loss for one company, over several years in the past:

```
BLOCK Blk_FinancialData
FIELDS
  Year: 1995..2020
  Turnover, Profit, Loss: REAL[6, 2]
RULES
  Year
  Turnover
  Profit
  Loss
ENDBLOCK { Blk_FinancialData }
```

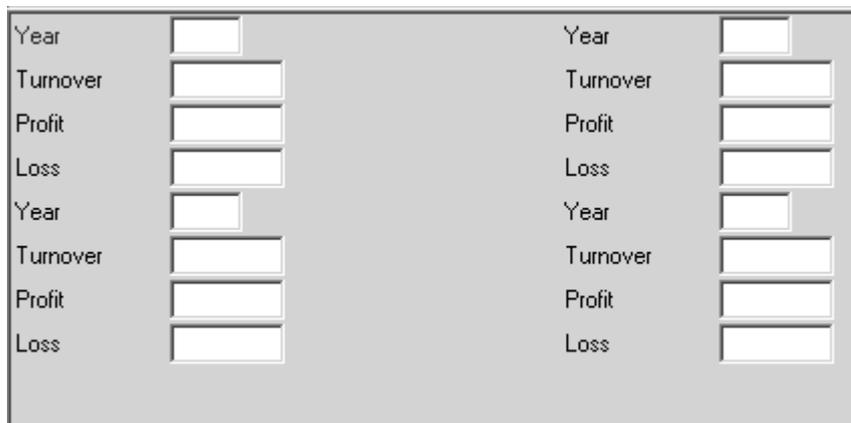
We use this block in another block:

```
BLOCK Blk_FinancialBlock
FIELDS
  SurveyYears: ARRAY[1..5] OF Blk_FinancialData
LOCALS
  I: INTEGER
RULES
  FOR I := 1 TO 5 DO
    SurveyYears[I]
  ENDDO
ENDBLOCK { FinancialBlock }
```

So there will be five sets of data, for five different years. The default grid is applied to the field, which is simply defined as

```
FIELDS
  FinancialField: Blk_FinancialBlock
```

With a standard mode library this will be shown as:



Year	<input type="text"/>	Year	<input type="text"/>
Turnover	<input type="text"/>	Turnover	<input type="text"/>
Profit	<input type="text"/>	Profit	<input type="text"/>
Loss	<input type="text"/>	Loss	<input type="text"/>
Year	<input type="text"/>	Year	<input type="text"/>
Turnover	<input type="text"/>	Turnover	<input type="text"/>
Profit	<input type="text"/>	Profit	<input type="text"/>
Loss	<input type="text"/>	Loss	<input type="text"/>

with the last block on another page.

The relevant part of the standard mode library is this:

Fill order:

Background:

Formpane border

Free Navigation

Sizes

View width:

View height:

Cell width:

Cell height:

Page width:

Page height:

The fill order in the default grid is vertical, the page width is 2 and the page height is 8. If, however, we change the fill order to horizontal, the page width to 4 and the page height to 5, we get a table-like presentation of the fields:

Year	<input type="text"/>	Turnover	<input type="text"/>	Profit	<input type="text"/>	Loss	<input type="text"/>
Year	<input type="text"/>	Turnover	<input type="text"/>	Profit	<input type="text"/>	Loss	<input type="text"/>
Year	<input type="text"/>	Turnover	<input type="text"/>	Profit	<input type="text"/>	Loss	<input type="text"/>
Year	<input type="text"/>	Turnover	<input type="text"/>	Profit	<input type="text"/>	Loss	<input type="text"/>
Year	<input type="text"/>	Turnover	<input type="text"/>	Profit	<input type="text"/>	Loss	<input type="text"/>

And if we change the fill order to vertical, the page width to 5 and the page height to 4, we get a kind of grouped presentation in which the years with their corresponding data are shown in vertical groups:

Year	<input type="text"/>								
Turnover	<input type="text"/>								
Profit	<input type="text"/>								
Loss	<input type="text"/>								

This paper was inspired by questions from the Statistics Netherlands Project “Joules In Motion” put to me by Nicolette de Bruijn and Tom Guldemond.

PRESENTING ANSWERS IN RANDOM ORDER.

A generic approach for presenting enumeration answers in random order in the Blaise Data Entry Program.

Marien Lina, Division of Business Statistics, Statistics Netherlands

1. Summary

A known methodological issue in survey research is structural response bias in case enumeration answers are presented in a fixed order. Respondents tend to select the first answers that are presented in an answer list. This occurs in self interviews such as internet surveys, but also in face to face surveys and telephone surveys. To minimize the amount of structural bias due to this undesirable response behaviour, answers may be presented in a random order.

There have been several attempts in the past to realize a random presentation order of answer categories in a Blaise questionnaire. Until recently, in Blaise this could only be done in a relatively complex way. The methods used in the old examples are not generic. They require modification for every new question. More generic solutions have been proposed but they require using the Blaise API, which is not available to everybody.

A new approach with Manipula in Blaise 4.8 makes it possible to realize this in a less elaborate way without using the Blaise API. This newly proposed method uses a Manipula setup and a generic include file for the Blaise data entry program. The Manipula procedures can be used in the data entry program relatively easy. For using the procedure a minimal amount of additional fields and instructions is required in the Blaise rules. This paper explains this generic approach to present answer categories in random order. A practical example will be added in the appendix.

2. Purpose

Structural bias appears for enumeration questions with a fixed order of presented answers. Presenting answers in a random order may help to avoid structural bias in measurement instruments. Respondents may prefer to respond a yes in stead of a no. Also there may be a tendency to take the answer category that has been offered first. When answers are presented in a fixed order, the answer that is presented first may cause a selective context and affect the attitude towards the second and the third answer category. If the answers are presented in a fixed order then respondents may be moved to the same context, which may lead to structural bias.

Different results may depend on the homogeneity and difficulty of the presented questions, the education level of the respondent, social desirability and context bias, the sensitivity and the subjectivity of the topic.

3. Used technical features

Various new and old technical features of Blaise have been combined to create a number of more or less generic procedures to display answer categories in random order.

3.1 Hiding enumeration categories

A characteristic feature used here is the option to hide categories of an enumeration when the descriptions of a category is empty. This can be used for a dummy enumeration question with a large number of answer categories. A limited number of answer categories can be displayed when the other categories have been left empty.

3.2 Calling Manipula procedures in the data entry program

Relatively new in Blaise is that there are methods for running manipula procedures in a Blaise data entry program. Results of Manipula procedures can be passed to the rules of the data entry program.

3.3 Retrieving meta information in Manipula.

Another relatively new option in Manipula is the retrieval of meta information. Meta information - for example the number of categories to be displayed - can be read in a manipula procedure, and used in a data entry program, as described above.

3.4 Compute and fix a random order of enumeration answers

In the rules of a Blaise datamodel, a presentation order can be computed. As soon as it is computed, the sequence can be fixed and it can be saved in the data, if replication of the same order of presenting answer categories is required.

3.5 Separate fields for asking and saving

A known trick for generating questions in a random order is to add a dummy question in a datamodel. Text fills can be used for the answers in the dummy questions. The array cells are always presented in the same order, but the text fills can be arranged to follow the random order as computed as described in 3.4.

3.6 A generic approach.

A generic include file allows you to use procedures for presenting categories in random order for enumeration questions for a variable number of answer categories. Any Blaise datamodel can include the generic include file, listed in appendix 2. The include file requires the availability of the prepared generic Manipula setup listed in appendix 3. The include file arranges that an enumeration question will be asked with a random presentation order of the possible answers. You will find a more detailed description below.

3.7 Specific modelib options.

The generic solution requires specific modelib options for the preparation of the datamodel that uses the include file. The generic procedure in this example uses a layout called *CatText*, using specific layout options that are listed in appendix 4.

To make it work, it is also necessary that, apart from the fields to be saved in the data, dummy array fields have been defined in the datamodel for asking the questions.

4. A generic solution

The example which is supplied in the appendix shows how you can combine the Manipula setup "shuffleanswers.man" (see appendix 3) with the generic include file "randinclude.bla" (see appendix 2) to use a random order for answer categories. The example works for enumeration questions with answer categories from 1 to 9. Code 0 will be treated as being empty. The values *dontknow* and *refusal* will be used and recognized. For recognizing empty as a value you will have to adapt the procedures in the include file and in the manipula setup. It is also quite simple to adapt the same example to make it work for a larger enumeration range, say from 1 to 97,

4.1 A simple datamodel.

Here is a small example datamodel that contains one enumeration question only.

```
DATAMODEL MyTranspo
TYPE
TTransport= (Walking, Bike, Car, Taxi, Bus, Train), dk, rf, empty
FIELDS { the question order is saved, the question order of the previous session is used!}
Transport "Can you tell hus how you got to work today?" : Ttransport
RULES
Transport
ENDMODEL
```

The answers in *Transport* are presented in a fixed order, as defined for *TTransport*.

4.2 Another simple datamodel.

To use the generic include file and the related manipula setup, some changes are required in the datamodel. Take notice that you should use specific modelib options (see appendix 4).

```
DATAMODEL MyRandomAnswers
TYPE
```

```

TTransport= (Walking, Bike, Car, Taxi, Bus, Train), dk, rf, empty
INCLUDE "RandInclude.BLA"
AUXFIELDS {The questions order is not saved. The order changes from session to session!}
  TransOrder : INTEGER { !! not saved in the data !! }
FIELDS { the question order is saved, the question order of the previous session is used!}
  Transport "Can you tell us how you got to work today?" : Ttransport
RULES
  Transport.KEEP
  TransOrder.KEEP
  ShuffleAnswers('Transport', TransOrder)
ENDMODEL

```

To arrange random answers for the question *Transport*, the actual question is not asked but instead it is kept in the rules:

```

RULES
  Transport.KEEP

```

The auxfield *Transorder* has been introduced to compute the random order of the answer categories. Take notice that this is an *AUXFIELD*. The presentation order will not be saved and computed again and again after re-opening the same form in the data entry program. If you would make it a *FIELD* then the order will be saved in the data and will stay the same after reopening the same data form.

The include file “randinclude.bla”, contains the procedure “*ShuffleAnswers*”. This procedure arranges the random presentation of the answers.

```

ShuffleAnswers('Transport', TransOrder)

```

The procedure contains a dummy enumeration question. The question text of the field *Transport* will be used in the dummy question. The answer categories will be copied from *Transport* to the dummy question, in the order as provided by the parameter *TransOrder*. Afterwards, the procedure returns the proper answer to the field *Transport*.

It is in this part where manipula procedures are called to retrieve meta information such as the question texts.

```

PROCEDURE GetQuestionText
PARAMETERS
  IMPORT FieldName: STRING
  EXPORT QuestionText : STRING
ALIEN('ShuffleAnswers.msu /KMyMeta=$dictionaryname', 'GetQuestionText')
ENDBLOCK

```

Other procedures ask for meta information, such as category names and the number of categories. These procedures are available generic include file and they are calling external procedures in the manipula setup. The files are listed in appendix 2 and 3.

4.3 Using the solution in other datamodels.

A more elaborate questionnaire of the previous questionnaire is available in appendix 1. It contains four enumeration questions that will present answers in a random order. For each enumeration question you want to have a random presentation order of the answers, like for the transport question, you will have to define a *AUXFIELD* or *FIELD*. This field is used as parameter for the procedure “shuffleanswers” in the include file. In the example above this was the *AUXFIELD TransOrder*. After that, the include file should be added in the datamodel, and the datamodel must be prepared with a specific modelib options (see appendix 4).

APPENDIX

Appendix 1 - An example Datamodel.

```
DATAMODEL MyRandomAnswers

AUXFIELDS
Hello "This example shows answer categories of enumeration fields in random order.
    @/Press Enter to start..." : (Enter), EMPTY
Goodbye "This completes the questionnaire. Press <Enter> to continue." : (Enter), empty
{ the answer descriptions are text fills of the array Answer[i] }

TYPE
TTransport= (Walking, Bike, Car, Taxi, Bus, Train, Balloon), dk, rf, empty
TWorldMap = (America, Europe, Asia, Australia, Africa, AntArctica), dk, rf, empty
TAnimal = (Tiger, Camel, Rabbit, Cow, Horse),dk, rf, empty
TYesNo = (Yes, No),dk, rf, empty

INCLUDE "RandInclude.BLA"

AUXFIELDS
TransOrder : INTEGER { !! not saved in the data !! }
CoffeeOrder : INTEGER
FIELDS
AnimalOrder: INTEGER { !! saved in the data !! }
WorldmapOrder : INTEGER
Transport "Can you tell us how you got to work today?" : TTransport
Animal "What is your favorit Animal?" : TAnimal
WorldMap "Where do you live?" : TWorldMap
Coffee "Would you drink coffee?" : TYesNo

RULES
Hello
Transport.KEEP
TransOrder.KEEP
ShuffleAnswers('Transport', TransOrder)
Animal.KEEP { these keep instructions are necessary }
AnimalOrder.KEEP { before calling the procedure }
ShuffleAnswers('Animal', AnimalOrder)
WorldMap.KEEP { these keep instructions are necessary }
WorldMapOrder.KEEP { before calling the procedure }
ShuffleAnswers('WorldMap', WorldMapOrder)
Coffee.KEEP { these keep instructions are necessary }
CoffeeOrder.KEEP { before calling the procedure }
ShuffleAnswers('Coffee', CoffeeOrder)
Goodbye
ENDMODEL
```

N.B. using SHOW in stead of KEEP may be helpful to understand what is happening in the data entry program.

Appendix 2 - Generic Include file.

```
TYPE
TTextArr = ARRAY[1..9] OF STRING

DummyEnum = (Item1 "^TempAnswer[1]",
             Item2 "^TempAnswer[2]",
             Item3 "^TempAnswer[3]",
             Item4 "^TempAnswer[4]",
             Item5 "^TempAnswer[5]",
```

```

Item6 ^^TempAnswer[6]",
Item7 ^^TempAnswer[7]",
Item8 ^^TempAnswer[8]",
Item9 ^^TempAnswer[9]"),dk,rf,empty

```

AUXFIELDS

```

PassScore : INTEGER, dk, rf, empty
QuestionText : STRING
Answer, TempAnswer : TTextArr
EnumSize : 1..9
dumdum : STRING

```

PROCEDURE GetQuestionText

```

PARAMETERS
  IMPORT FieldName: STRING
  EXPORT QuestionText : STRING
ALIEN('ShuffleAnswers.msu /KMyMeta=$dictionaryname', 'GetQuestionText')
ENDBLOCK

```

PROCEDURE GetSize

```

PARAMETERS
  IMPORT FieldName1: STRING
  EXPORT EnumSize : INTEGER
ALIEN('ShuffleAnswers.msu /KMyMeta=$dictionaryname', 'TellSize')
ENDBLOCK

```

PROCEDURE GetOrd

```

PARAMETERS
  IMPORT FieldName1: STRING
  EXPORT OrdVal : STRING
ALIEN('ShuffleAnswers.msu /KMyMeta=$dictionaryname', 'GetOrd')
ENDBLOCK

```

PROCEDURE GetCatNameI

```

PARAMETERS
  IMPORT FieldName1: STRING
  IMPORT XVal : STRING
  EXPORT A1 : STRING
ALIEN('ShuffleAnswers.msu /KMyMeta=$dictionaryname', 'GetCatnameI')
ENDBLOCK

```

PROCEDURE GetRandomSequence { this procedure sets the random order of categories for one question }

```

PARAMETERS
  IMPORT Size : INTEGER { the number of categories in the enumeration }
  EXPORT Order: INTEGER { a number that holds the sequence of the questions }
AUXFIELDS
  Stringeling : ARRAY[1..9] OF INTEGER
  RandomSet: ARRAY[1..9] OF INTEGER { random numbers used to sort stringeling }
  DNum : integer
  ENUMSIZE : 1..9
LOCALS I,J: INTEGER
RULES
  EnumSize:=Size
  FOR I:= 1 to EnumSize DO
    Stringeling[i]:= i { initial categories are sorted to ORD value }
    RandomSet[i]:= random(9999999999999999) { high number to minimize double numbers }
  ENDDO
  FOR i:= 1 to 8 DO { sort the array to random numbers }
    FOR j:= 2 to 9 DO
      if (i<j) AND (j<=enumsz) THEN
        IF RandomSet[j]>RandomSet[i] THEN
          Stringeling.exchange(j,i)
          RandomSet.exchange(j,i)
        ENDIF
      ENDIF
    ENDDO
  ENDDO

```

```

Dnum:= 0 { represent the order as a number }
j:=1
FOR i:= 1 TO 9 DO
  IF i<= enumsize THEN
    DNum:= DNum + j*stringeling[i]
    j:= j * 10
  ENDIF
ENDDO
Order:= DNum
ENDPROCEDURE

PROCEDURE RandomAsk { Use a dummy field of an enumeration type to ask the questions and }
  { display the proper question and answer texts with the question }
PARAMETERS
  IMPORT Desc: STRING
  TRANSIT D : integer { import the previously selected category, export the selected categorie in the procedure }
  IMPORT Order:integer { import the order of the answers }
AUXFIELDS
  AskIt ""QuestionText" / ""DESC" : DummyEnum
  MyOrder : String
  MyAnswer: ARRAY[1..9] of string
  Stringeling : ARRAY[1..9] OF 1..9
LOCALS
  i : INTEGER
  j : INTEGER
RULES
  MyOrder:= Str(Order)
  FOR i:= 1 to EnumSize DO
    Stringeling[i]:= val(substring(Myorder,i,1))
  ENDDO
  FOR i:= 1 to 9 DO
    IF i<=EnumSize THEN
      TempAnswer[i]:= Answer[Stringeling[i]]
    ELSE
      TempAnswer[i]:= ""
    ENDIF
  ENDDO
  AskIt.KEEP
  IF AskIt=EMPTY THEN
    IF D= 10 THEN
      ASKIT:=DK
    ELSEIF D= 11 THEN
      ASKIT:=RF
    ELSE
      FOR i:= 1 to EnumSize DO
        IF D = val(substring(Myorder,i,1)) THEN
          AskIt:= i
        ENDIF
      ENDDO
    ENDIF
  ENDIF
  IF AskIt = RESPONSE THEN
    D:= Stringeling[ORD(AskIt)]
  ELSEIF AskIt= DONTKNOW THEN
    D:= 10
  ELSEIF AskIt= REFUSAL THEN
    D:= 11
  ELSE
    D:= 0
  ENDIF
LAYOUT
  At ASKIT fieldpane CatText
ENDPROCEDURE

PROCEDURE GetMeta
PARAMETERS IMPORT MetaName:STRING

```

```

LOCALS counter:integer
RULES
  GetQuestionText(MetaName, QuestionText)
  GetSize(MetaName, EnumSize)
  FOR Counter:= 1 to 9 DO { assigning the names of the categories }
    GetCatName(MetaName, STR(Counter), Answer[Counter])
  ENDDO
ENDPROCEDURE

PROCEDURE UnShuffle
PARAMETERS
  IMPORT FName : STRING
  IMPORT Score : INTEGER
ALIEN('ShuffleAnswers.msu /KMyMeta=$dictionaryname', 'UnShuffle')
ENDBLOCK

PROCEDURE ShuffleAnswers
PARAMETERS
  IMPORT MetaName:STRING
  TRANSIT ItemOrder: INTEGER
LOCALS
  OrdVal: STRING
RULES
  GetMeta(MetaName)
  IF ItemOrder=EMPTY THEN
    GetRandomSequence(EnumSize, ItemOrder) { computing the random sequence if still empty }
  ENDIF
  GetOrd(MetaName, dumdum)
  PassScore:= val(dumdum)
  RandomAsk(MetaName, PassScore, ItemOrder)
  UnShuffle(MetaName, PassScore)
ENDPROCEDURE

```

Appendix 3 - Generic Manipula Setup

```

PROCESS TestMeta
USES MyMeta (VAR)
TEMPORARYFILE x:MyMeta
SETTINGS INTERCHANGE=SHARED

PROCEDURE UnShuffle { put answered value "Score" back to field "FName" }
PARAMETERS
  IMPORT FName : STRING
  IMPORT Score : STRING
INSTRUCTIONS
  IF val(Score) = 10 THEN
    x.PUTVALUE(FName, DONTKNOW) { take into account DK values }
  ELSEIF val(Score) = 11 THEN
    x.PUTVALUE(FName, REFUSAL) { take into account RF values }
  ELSEIF val(Score) >0 THEN
    x.PUTVALUE(FName, Score)
  ELSE
    x.PUTVALUE(FName,EMPTY)
  ENDIF
ENDPROCEDURE

PROCEDURE GetOrd { Read the current value in field "FieldName1" }
PARAMETERS
  IMPORT FieldName1 : STRING
  EXPORT OrdVal : STRING
AUXFIELDS
  FieldStat : STRING
INSTRUCTIONS
  FieldStat:= x.GETFIELDINFO(FieldName1, 'FIELDSTATUS')
  IF FieldStat = 'DONTKNOW' THEN
    Ordval:= '10' { take DK and RF values into account }
  ELSEIF FieldStat = 'REFUSAL' THEN

```

```

    Ordval:= '11'
ELSEIF FieldStat = 'EMPTY' THEN
    Ordval:= '0'
ELSE
    OrdVal:= x.GETVALUE(Fieldname1,UNFORMATTED)
ENDIF
ENDPROCEDURE

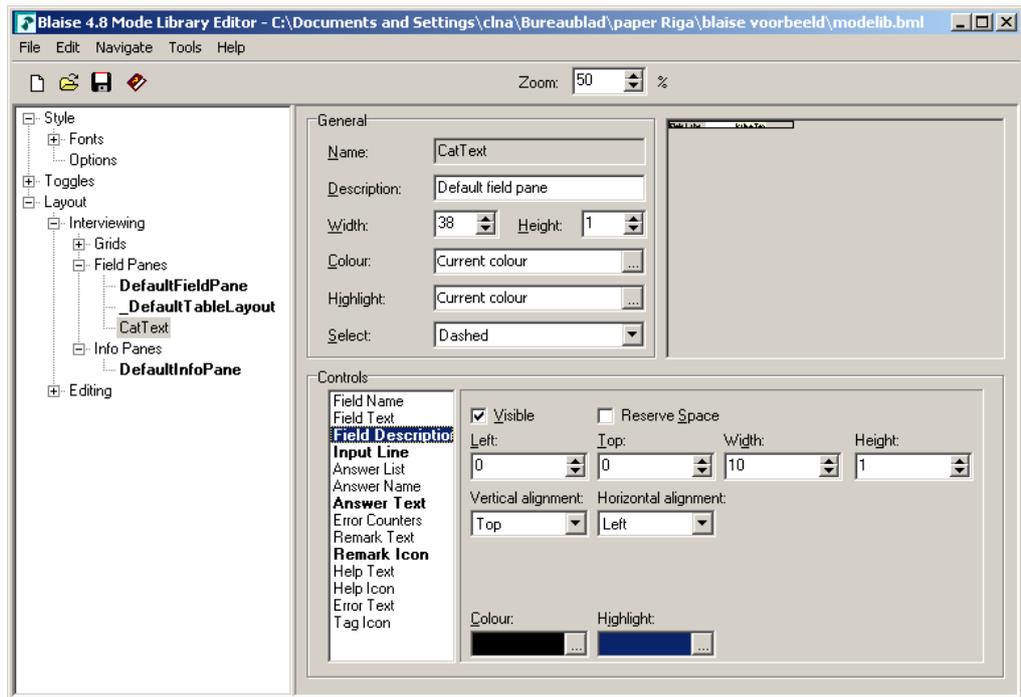
PROCEDURE TellSize { how many categories are in the enumeration }
PARAMETERS
    IMPORT FieldName1 : STRING
    EXPORT Size : STRING
AUXFIELDS
    ix : integer
INSTRUCTIONS
    ix:= val(x.GETFIELDINFO(FieldName1,'CATEGORIES.COUNT'))
    Size:= STR(ix)
ENDPROCEDURE

PROCEDURE GetCatName1 { returns the category name of FieldName[XVal]}
PARAMETERS
    IMPORT FieldName1 : STRING
    IMPORT XVal : STRING
    EXPORT A1 : STRING
AUXFIELDS
    ix : INTEGER
INSTRUCTIONS
    ix:= val(x.GETFIELDINFO(FieldName1,'CATEGORIES.COUNT'))
    IF VAL(XVAL)<= IX THEN
        A1:= x.GETFIELDINFO(FieldName1,'CATEGORIES[' + XVAL + '].NAME')
    ELSE
        A1:= "
    ENDIF
ENDPROCEDURE

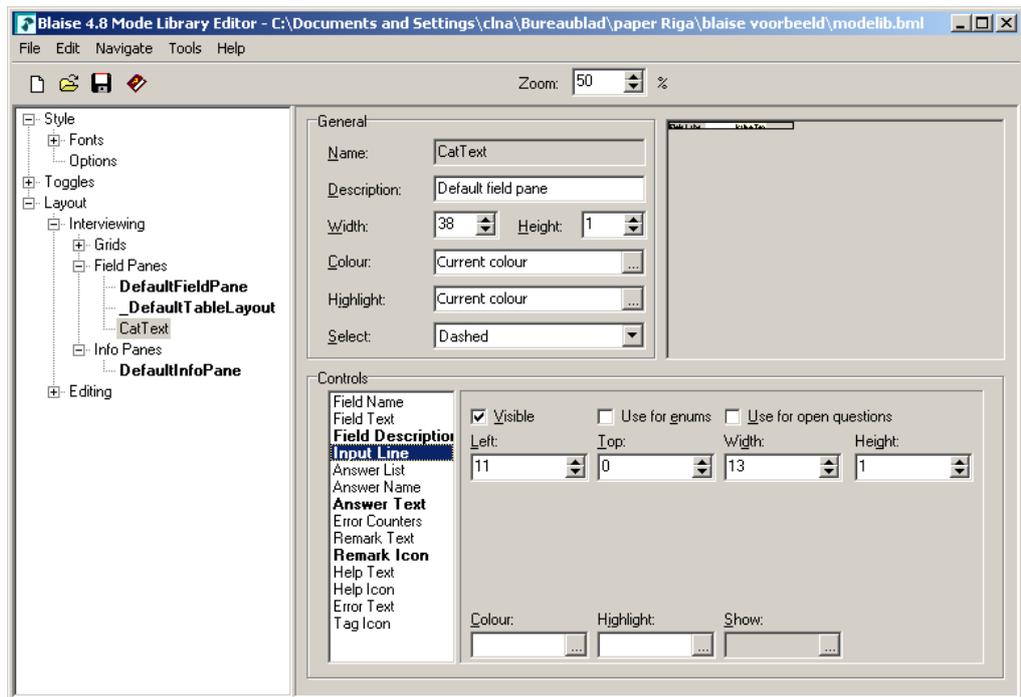
PROCEDURE GetQuestionText { returns the questiontext of FieldName }
PARAMETERS
    IMPORT FieldName: STRING
    EXPORT QuestionText : STRING
INSTRUCTIONS
    QuestionText:= x.GETFIELDINFO(FieldName,'QUESTIONTEXT')
ENDPROCEDURE

```

Appendix 4 – Used Modelib Settings



The Field Pane *CatText* is used in the generic include file for presenting and asking the dummy question with the random order of the answer categories of the enumeration.



The figures show the settings for displaying the *Field Description*, *Input Line* and *Answer Text* for the Field Pane *CatText*. The Remark icon is in the default position.

