GENERATING MULTIPLE VERSIONS OF QUESTIONNAIRES

Mark Pierzchala
National Agricultural Statistics Service, Fairfax, USA

1. Challenges of versions of questionnaires

Any system that is to be considered for data collection and editing in the National Agricultural Statistics Service (NASS) must be able to handle versions of questionnaires. NASS is decentralized with headquarters in Washington DC and 45 state offices where most of the production work is conducted. For some surveys, there are potentially 45 versions of questionnaires that must be accommodated. The idea of generating versions of interactive editing and interviewing instruments by computer came from Wouter Keller of the Netherlands Central Bureau of Statistics. This paper presents the results of preliminary work done in NASS. It has been shown that a computer program can generate versions of instruments easily. Computer generated instruments have not yet been used in production.

2. Motivation for automated generation of versions

The Computer Assisted Survey Section in NASS has just been given a mandate to produce interactive processing instruments for editing, data entry, and data collection for all of NASS's surveys. This mandate is much expanded from just a few months ago when the section was called the CATI section. The CATI section concentrated just on Computer Assisted Telephone Interviewing for a subset of NASS surveys. With the expanded mandate, and with only a small increase in personnel in that section, NASS must re-evaluate its instrument production techniques. While it is possible to manually program instruments for versions of surveys the cost is very high. The cost must be measured in terms of both direct costs and opportunity costs. Direct costs involve time needed to program and maintain the versions of instruments.

Opportunity costs may overwhelm direct costs. These involve the implementation that is not carried out or is carried out on a delayed schedule because production and maintenance of the first instruments is so time consuming. By using an integrated system such as Blaise it is possible to reduce costs by programming one instrument for editing and interviewing. This represents a savings of about 50% over programming one system for editing and one for interviewing. This improvement is not good enough. It is also important to keep in mind the huge United States deficit which is about 400 billion dollars. Increased financial pressure upon NASS and other agencies is a certainty. The broader challenge will be to produce the same (or better) product in the future with fewer resources. By aggressive and intelligent use of technology this challenge can be met.

3. Example of versions of questionnaires

The Quarterly Agricultural Survey (QAS) best illustrates the challenge of multiple versions in NASS. Within each QAS questionnaire are several standard sections such as crops, grain stocks, hogs, and chickens. The crops sections of the December QAS versions offer the most extreme example of the challenge. The crops sections in the December quarter of the QAS will survey different crops in the various states as demonstrated by the three fictitious and simplified lists below:

STATE A	STATE B	STATE C
Corn	Irrigated Corn	Corn
Soybeans	Non-Irrigated Corn	Soybeans
Potatoes	Irrigated Sorghum	Alfalfa Hay
Sorghum	Non-Irrigated Sorghum	Sunflowers
	Peanuts	Tobacco
	Potatoes	
	Rice	
	Irrigated Soybeans	
	Non-Irrigated Soybeans	
	Cotton	

Several aspects of the version challenge are apparent:

1. there are different crops in different states,
2. there are different approaches to surveying a crop, especially in some states it is necessary to distinguish between irrigated and non-irrigated crops while in other states no distinction is made, and
3. the order of the crops differs from state to state.

Other aspects of the version challenge that are not as apparent include differences in question text, units of production (e.g., pounds, tons, or bushels), SAS variable names, and unique item codes that have to be associated with each question.

Various items of information are gathered about each kind of crop. For example for corn the items for each state above are:

CORN, STATES A & C	CORN, STATE B
acres planted,	irrigated acres planted
acres harvested for grain,	irrigated acres harvested for grain
production of grain,	irrigated production of grain
acres harvested for silage,	irrigated acres harvested for silage
production of silage,	irrigated production of silage
other corn acres.	irrigated corn acres for other uses
	non-irrigated acres planted
	non-irrigated acres harvested for grain
	non-irrigated production of grain
	non-irrigated acres harvested for silage
	non-irrigated production of silage
	non-irrigated corn acres for other uses

4. Building upon similar structures

There are similar structures between and within crops that can be used to simplify the task of generating versions. For example, the following structure, as exemplified by the following ROUTE statement, can be applied both to corn and to sorghum as well as other crops.

```
ROUTE    {consolidation of route statements from three blocks}

   Planted;                  {irrigated, non-irrigated, or both}
   If Planted > 0 then
     Harvestd                 {nested grain block}
     If Harvestd > 0 then
       Prodction              {nested grain block}
     endif;
     Harvestd                 {nested silage block}
     If Harvestd > 0 then
       Prodction              {nested silage block}
     endif;
   endif; {planted}
   OtherUse;
```

A simpler structure that can be applied to several crops such as soybeans, potatoes, and tobacco is:

```
ROUTE                         {all one block}
  Planted
  If Planted > 0 then
    Harvestd                  {one use of crop only}
    If Harvestd > 0 then
      Prodction
    endif;
  endif;
```

Edits in the CHECK and SIGNAL paragraphs have similar form between crops and states, even if details differ. For example:

```
SIGNAL
((softlow < Prodction/Harvestd) and (Prodction/Harvestd < softhigh))
```

Notice that no matter what the crop (and sometimes even twice in the same crop), the same questions names are used. This is deliberate as this will make excellent use of blocks (and nested blocks) in Blaise and also of its "dot notation" and significantly cut down on the coding and maintenance of the instruments. Also note that the SIGNAL paragraph has edit limits that are defined with variables, not hard-coded numbers. This allows dynamic definition of edit limits between crops. The idea is to build upon similarities between versions of questionnaires, not to agonize over differences.

5. Breaking down the version challenge into manageable parts

The strategy used in generating versions of instruments is to separate aspects of programming that change from crop to crop from those that do not change from crop to crop. In Blaise this separation is made very neatly. The things that do change from crop to crop appear mostly in the QUEST paragraph, while the things that stay the same appear totally in the ROUTE, SIGNAL, and CHECK paragraphs. Thus the problem is simplified if a way can be found to apply the same ROUTE, SIGNAL, and CHECK paragraphs to different QUEST paragraphs. Conceptually this may be done in one of two ways:

1. create a generic QUEST paragraph that is dynamically redefined for each crop, or
2. create a different text file for each crop's QUEST paragraph.

The first option is mostly but not totally possible in Blaise. It would require many expensive text substitutions and an external file that would hold the text. Also Blaise does not allow some parameters to be dynamically defined such as question numbers (item codes in NASS) and question labels. For these reasons and others, this demonstration has been constructed using the second method. Examples of QUEST paragraphs are as follows:

```
QUEST  {for corn}
  Planted  "/100/  How many acres of SOYBEANS were planted for all
   purposes?" "CSYB__PL" : 0..997;
  Harvestd "/101/  How many acres of the $PLANTED acres for SOYBEANS
   were harvested?" "CSYB__HV" : 0..997;
  Prdction "/102/  What was the production of SOYBEANS in BUSHELS?"
   "CSYBPROD" : 0..9999997;

QUEST  {for tobacco}
  Planted  "/200/  How many acres of TOBACCO were planted?"
  "CTOB__PL" : 0..99.7;
  Harvestd "/201/ How many acres of the $PLANTED acres for TOBACCO
   were harvested?" "CTOB__HV" : 0..99.7;
  Prdction "/202/ What was the production of TOBACCO in POUNDS?"
   "CTOBPROD" : 0..9999997;
```

About the only thing that is the same between the two QUEST paragraphs are the question names. The item codes, the SAS variable names, the units of production, the valid values, and even some of the question text differ. However, the same ROUTE, SIGNAL, and CHECK paragraphs can be applied to both QUEST paragraphs. This is done by use of blocks in Blaise and by use of INCLUDE statements as illustrated below:

```
BLOCK crn;
  QUEST {for corn}
  INCLUDE "RSC1.fle";  {holds ROUTE, SIGNAL, and CHECK paragraphs}
ENDBLOCK;
QUEST  corn : crn;

BLOCK tob;
  QUEST {for tobacco}
  INCLUDE "RSC1.fle";  {holds ROUTE, SIGNAL, and CHECK paragraphs}
ENDBLOCK;
QUEST  tobacco : tob;
```

The INCLUDE statement is used here to repeat the same code twice. The reason that Blaise does not confuse questions between the two blocks is because of the dot notation that Blaise uses. For example the names of the planted acres questions defined above are CORN.PLANTED and TOBACCO.PLANTED.

6. Customization of blocks

It remains to customize each block as regards edit limits (and perhaps a few other things as well). This customization is done in the ROUTE paragraph at the instrument level (highest level of organization) as follows:

```
ROUTE                                      {instrument or questionnaire level}
  ID;                                                  {Identification block}
  Land;                                                 {some land questions}

                                                {compute edit limits for corn}

    Compute softlow := 50; Compute softhigh := 150;
    Compute hardlow := 0;  Compute hardhigh := 300;
  Corn;                                                 {call corn questions}

      {compute edit limits for tobacco}
    Compute softlow := 150; Compute softhigh := 450;
    Compute hardlow := 0;  Compute hardhigh := 800;
  Tobacco;                                           {call tobacco questions}
ENDQUEST.
```

In practice, the variable edit limits are calculated from an external file as they change from state to state for the same crop. The code above is a simplified extract of what one version of an instrument would look like. It can be programmed by people but there is no reason that major parts of it cannot be automatically generated. Remember that there are potentially 45 versions of the questionnaire to produce. Manual coding would be tedious, time-consuming, and subject to error. Even though this method is better than current methods used with another system in NASS, it can still be improved.

7. Specialization of tasks

The nice thing about the automated approach is that in separating the QUEST paragraphs from the ROUTE, SIGNAL, and CHECK paragraphs, different people can build and maintain them. For example, the QUEST files can be the responsibility of secretarial and clerical staff because they are simply text files of set and structured format. The ROUTE, SIGNAL, and CHECK paragraphs can be built and maintained by subject matter specialists.

8. Requirements for automatic generation of versions

Three major requirements for automatically generating versions of instruments are a library of code, a parameter file of specifications, and a program that generates the versions of instruments. The library serves as a resource of coded segments that the generator program assembles according to the parameter file of specifications.

9. Library of code

The concept of a library is very important. By breaking up the Blaise coded segments as described there are potentially hundreds of small files where before there were just several larger ones. They cannot all be placed in one directory. They must be placed in a simple subdirectory tree whose structure is readily readable and understandable to those who must work with it. Also, the file names must conform to a naming convention which is also understandable. The directory tree which makes up the library is displayed:

```
BLIB    CROP1                      {1st quarter crops}
        CROP2                      {2nd quarter crops}
        CROP3                      {3rd quarter crops}
        CROP4     CORN             {4th quarter corn}
                  SORGHUM          {4th quarter sorghum}
                  etc.

        R_S_C                      {Route, Signal, Check paragraphs}
        HOGS
        STOCKS
        etc.
```

BLIB stands for Blaise LIBrary and R_S_C for Route, Signal, and Check. Within a subdirectory such as CORN there are potentially several files of QUEST paragraphs, one for each way that corn may be surveyed across the country in the December quarter. File names must describe (even if cryptically) what they contain. The first part of the two part file name describes the crop and the approach to surveying it. The extension of the file name will be used to designate a file that contains minor modifications to one of the general survey approaches if required in a particular state. For example:

```
CORN1G
CORN1S
CORN1G.55
```

CORN1G holds a QUEST paragraph concerning approach 1 of surveying corn for just the Grain questions. CORN1S contains similar questions for Silage. These two files are in fact treated as the default (or master) files for corn that would normally be used in all appropriate states. However if state 55 (Wisconsin) needs different wording or range of valid values (but not different question names, item codes, or SAS variable names) then CORN1G is copied to CORN1G.55 and the latter file is modified. The version generating program would then choose CORN1G.55 over CORN1G when generating the version for state 55.

For each structure that is found in the QUEST paragraphs there must exist a corresponding R_S_C file that sets the ROUTE, SIGNAL, and CHECK paragraphs. For example, files CORN1G and SORG1G (corn and sorghum respectively) have the same structure (but different details). A file found in the R_S_C directory that would apply to both CORN1G and SORG1G would have a descriptive name such as ROW1G.FLE which would stand for "ROW crop 1 for Grain". The extension here has no special meaning but serves as a convenient way for searching for certain kinds of files in the library. While there are many QUEST files for crops there are only several R_S_C files because relatively few structures are needed to cover all the crops.

10. Parameter file of specifications

The parameter file(s) of specifications must contain pertinent information for each state. Ideally there would be one file where each state's specification would be contained on one line. The specification file can be either an ASCII file or a Blaise file. The former might be generated from the agency's specification system, the latter from a subsidiary

Blaise instrument especially designed for hand entry of specifications. NASS's current file of specifications (used for SAS editing and summary) falls a little short of what is needed for the automatic generation of Blaise instruments. Each state's specification must contain a list of valid item codes (which is equivalent to stating which crops are valid for that state), SAS variable names, the order in which crops are to appear, and the state's ID number. It may be necessary to have most of the information in an ASCII file and other information such as order of crops designated in a subsidiary Blaise instrument. If a subsidiary Blaise specification instrument is necessary then it would be designed so that a clerk could fill in the proper information.

11. Generator program

The program that generates the versions from the library of code and the parameter file need not be very complicated. It needs to be able to read information from the parameter file and draw upon the code from the library. The generator program can be written either in Manipula (a Blaise file manipulation utility) or in a programming language such as Turbo Pascal. All work so far has been done with Manipula. It is able to read either ASCII or Blaise files directly and to write text files (including DOS BAT files) based on the information in the parameter file. The sequence of events is as follows:
- The generator program reads a line from the parameter file.
- Lines are written into a text file by the generator program. The lines in the text file refer to code from the library.
- The Blaise program is built up from a standard front part, the text file of library references, and a standard ending part.
- The Blaise program is syntax checked and compiled from DOS.
- The compiled program is copied into the proper holding directory.

The lines that are written into the text file are mostly INCLUDE and QUEST statements. A portion of such a program is as follows:

```
INCLUDE '\blib\crop4\corn\corn1g';  {corn for grain questions}
INCLUDE '\blib\R_S_C\row1a.fle';    {route and edits}
QUEST   grain : firstuse;           {end of first nested block}
INCLUDE '\blib\crop4\corn\corn1s';  {corn for silage questions}
INCLUDE '\blib\R_S_C\row1a.fle';    {route and edits}
QUEST   silage : secnduse;          {end of second nested block}
INCLUDE '\blib\R_S_C\row1.fle';     {end of whole block}
```

These lines represent the code from the library that will ask about 10 questions, route them properly, and apply approximately 10 edits.

The person who writes the generator program does not have to be a systems person but should have some facility with computers. The generator program must be written in a very robust way. It should have capability to add new crops and new sections of the questionnaire without alteration. In fact, once written, it should apply to all quarters or periods of a particular survey and run for several years without maintenance. NASS does not want to trade a problem of maintaining instruments to one of maintaining a generator program.

The generation of versions of instruments is done in batch, one version after another. Best results are obtained with a powerful computer (486 processor or above) with enough RAM and extended memory to hold the library of code, the Blaise system files, Turbo Pascal, and work space for the syntax check and compilation. Probably 16 Mb would be sufficient. Time needed to generate each version is approximately 1 minute. Computers with this configuration are now starting to appear on desks in NASS. The only time the hard disk is used is in storing the compiled instruments.

If problems are found after the versions of instruments are generated, then repairs are not made to the instruments, but to the library of code or to the generator program. After repairs are made there, appropriate versions are regenerated.

12. Survey management

Survey management is standardized across state offices and across versions by use of standard front part of the instruments that includes a block of code for management purposes. This management block contains survey control questions, many of them with the selector attribute. Management tables in Abacus or Manipula are applied across versions of instruments and even across different surveys. Each state produces the same tables and accesses or moves forms (with the Blaise forms manager) with the same selector questions. The only difference between the versions of the instruments is the subject matter.

13. Instrument administration

There has been some question concerning the administration of many different versions of instruments from headquarters. This need not be more difficult than current practice and should even be easier given proper management. The idea is to build the instruments correctly in the first place and avoid having to repair them in mid-survey. Some survey administration could be automated. For example, a computer program could distribute instruments to the correct location. A database of information about the instruments already exists in the parameter file of specifications. This information can also be used to help administer the survey. For example, if there is a problem with a block of code that concerns only 4 states, then the database could be used to designate which 4 states are concerned. Repairs are posted only to the 4 states, the other 41 states are not bothered. In the current system if there is a repair to be made then all states must update their instruments, even if it does not concern their survey program.

Testing different versions of an instrument is an issue. The question is how much testing is needed. The key is not to wait until versions of instruments are automatically generated before testing. The time to test is before chunks of Blaise code are put in the library. The library should consist only of tested code, so when these chunks are pulled into an instrument they should be regarded as tested subroutines. However, there are still things that might go wrong. For example, some of the crops may not have been placed in a version of the instrument, or a few edits that go across crops may not work. The Blaise setup generator will help with some of this testing. It can produce reports of item codes and SAS variable names (with NASS customized setups) for each version of the instrument. These can be manually inspected, however a computer comparison of the reports against the original specification file would be preferred. The dictionary provided by Blaise as well as the technical description will be very useful as well. The testing that remains should be carried out according to a strict protocol and may be done by clerical personnel. Inevitably the final testers will be the users in the state offices as is done now. One point to keep in mind is that if instruments are generated automatically then there is more time to test them afterwards but less need to do so.

14. Multiple versions versus one massive program

Some people wonder why multiple versions of an instrument should be generated. Why not produce one massive instrument that can handle all situations and use the parameter file of specifications to determine which questions to ask in which state? In fact this can be done. The computer generation technique need not produce many versions of one instrument. The approach can be used to produce one instrument that invokes the right questions in the right state. However there are two strong reasons for generating multiple versions. First is that Blaise is a forms based system with two modes of operation. The interviewer should not have to see many or any inappropriate questions in the forms (bottom) part of the screen even if they are not invoked. It is possible to use the NEWPAGE question in order to keep inappropriate questions from enumerator sight, but this makes less efficient use of the form, adding pages unnecessarily. The problem becomes worse in the editing (CADI) mode where the routing is passively enforced, not dynamically controlled. The data editor would have to deal with up to three times as many pages, many of them unfilled, as would be necessary in a customized instrument.

The second reason that individual versions are preferred relates to unusual aspects of NASS's survey population, United States farms. The sampling universe is relatively small and getting smaller, and farm operations can be very large and diversified. As a result some farms are surveyed several times during the year for one survey or another. This situation worsens every year. As several surveys are conducted at any one time it is desirable to contact the farmer one time (within some period of time) and to ask all appropriate questions for all appropriate surveys in one session. Currently this means that the first survey is administered in CATI but then the interviewer leaves CATI and administers the rest of the questions on paper. For the second and subsequent surveys the benefits of CATI are lost for that farmer and the data must then be entered into the system causing more work. A solution to the multi-survey problem is to construct instruments that can handle multiple surveys. By customizing versions for each state, the size of each instrument is kept manageably small allowing more modules to be added for other surveys. A parameter file is used to control the interview and invoke the correct surveys in the instrument. This method should reduce respondent burden by asking certain questions only once. For example, if the name and address information is verified for one survey in the interview, then it will not be necessary to verify it for other surveys in the same interview. The handling of the data after data collection would be facilitated by the use of subfiles available in Blaise. Each survey's data will be kept in its own set of subfiles. A powerful feature of Blaise is that survey management can be carried out at the subfile level in addition to the management at the form level. The generation of MEGA-VERSIONS of interviewing instruments is a straight forward extension of generation of versions of instruments as covered in this paper. However there are many interesting ramifications as regards data handling and survey management. These will be the subject of a future paper.

15. Improvements in Blaise

There are a few enhancements to the Blaise system that would help in the generation of versions of instruments. These include the capability of passing question numbers and question labels as parameters, allowing question numbers to be used in tables, nesting of an INCLUDE within another INCLUDE (two levels only would be sufficient), and a new kind of question similar to NEWPAGE called NEWCOL. The NEWCOL question would help with formatting the screen between the CATI/CAPI mode and the CADI mode when the columnar format for questions is used. It would start the next question at the top of the column. Currently it is difficult to make the screen look as nice as possible in both modes. None of these enhancements is critical to the success of this method of generating instruments, they would just make life easier.