Whatever Happened To Our Data Model?
Documenting Change in Continuous Surveys.

Sven Sjödin - National Centre for Social Research, UK

Many, if not most, of the large social surveys are either conducted continuously or are repeated at fixed intervals. The survey processing systems can usually be recycled with some alterations for the next wave. The effort spent on these amendments depends on how much the data model has changed. The amount of work can, however, be reduced by good documentation of the data model changes.

Of course, there are already methods to document data collection instruments - survey organisations have a strong business need for such documentation. Moreover, the TADEQ project will develop a standard system for documenting computer assisted interviewing instruments once it is completed. Meanwhile, most organisations have developed their own systems for documentation. Others have to make do with what the Blaise system can offer: the Blaise data model specification, the Structure Viewer and Cameleon. However, while some of these documentation systems may be extremely elaborate they fail to address the issue of alterations to data models, they still only give the static picture of a data model.

1. The Survey

Our need to develop a system for documenting changes between successive data models arose from the Family Resources Survey. This survey has a very complex instrument, as it is designed to collect information on all kinds of household income in great detail. The survey is conducted in co-operation between the National Centre for Social Research and the Office for National Statistics (ONS). The data collection and editing are shared between the two organisations, while the National Centre maintains the data model and the ONS processes the data for delivery. The data collection is conducted continuously and the data model is revised once a year.

The Department of Social Security (DSS) - the client - has specified the delivery data as a set of tables together with detailed meta data documentation. The production of the delivery tables requires a very complex data processing system. Even small changes to the data collection instrument will have consequences for the processing of the data. For example, an added answer category to catch a new benefit payment requires a new output table entry with a unique key value. The data output system is too elaborate to re-design each year. To make the necessary amendments, the specialists at the ONS and the DSS need the best possible documentation of what exactly has changed since the previous wave.

2. Program Requirements

There are four key requirements for the documentation of changes between successive data models:

- It must be generated from the Blaise files, either the data model specification or the prepared data model, as manually maintained documents may not reflect the true contents of the data models;
- The documentation must record all the relevant changes without over-reporting irrelevant differences;
- It should be automated as far as possible; and
- Finally, the system should be applicable to all surveys.

2.1 Comparing What?

Which is the best basis for the comparison: the data model specification or the prepared data model?

The data model specification is the complete definition of the data model, often referred to as the source code. It is possible to compare two data model specifications and report on the differences using a standard file compare utility, e.g. FC or DIFF. However, this method would generate a vast output file. Every differing byte in the two input files is reported. It requires a parser to analyse the output file and eliminate the irrelevant differences. At this level of program development, it would be more

efficient to build a parser that produces a standardised description of each data model specification and then to compare the two descriptions.

The alternative to the data model specification is the prepared data model. This is not directly readable. It can only be accessed via the Data Entry Program, the Structure Viewer or Cameleon. Cameleon has a well-defined language that allows you to specify how to record information about a data model in an output file. All the information from the FIELDS sections is available for the output specification. However, Cameleon has no knowledge of the RULES sections.

2.2 No Rules?

From the point of view of the current users, reporting differences in the RULES sections between successive data models may not be a necessary feature as long as the document signals changes to the number of block and field iterations. For example, in a household survey instrument, the number of possible respondents has been reduced from twelve to ten. This is an important change that must be reported, but it doesn't require any direct knowledge of the RULES, because the change will also be reflected in the array sizes of person level blocks and fields.

Another RULES related change that may be of interest is in the sequencing of the fields. Although the order of the fields in the FIELDS section doesn't have to be the same as the order in the RULES section, it is good programming practice to synchronise them; and Cameleon can be instructed to assign a sequence number to each field.

3. The Design

In designing the system to document changes, the basic idea is to let Cameleon record a standardised set of properties for each field, then sort them by field name and compare the lists. Any reported difference is categorised into one of three types;

- A deletion - a field or answer category that has been removed from the new data model;
- An insertion - a new field or answer category; or
- A change - a field that is re-specified in some way.

3.1 The Level of Cameleon Output

The implementation of this simple idea of comparing standardised and sorted lists of fields is not as easy as it may seem. There are a number of factors to consider:

- **First, how to handle the answer categories for enumerated and set field types?** The documentation system has to be able to detect and record any changes to single answer categories, even if the field level properties remain unchanged. The answer categories are on a relational level below the field to which they belong. Each field can have a number of answer categories, but each answer category can only belong to one field. The easiest solution is to let each answer category form a separate record in the output file so that they share the field level properties but differ in the answer level properties. The resulting file will then have the answer category, rather than the field, as its basic level.

- **Second, whether to record all the instances of a field or just one instance?** This issue arises out of the fact that fields, and blocks of fields, can be repeated in arrays. Recording all the instances of a field would give a much larger output file and, unless suppressed, changes will be reported over and over again for every instance of such a field. Selecting just one instance of a field is more efficient, as long as a change to the array size can be detected.

- **Third, how to deal with different fields that share the same field name?** Blaise allows fields to have the same name as long as they are declared in different blocks. As the Cameleon output is sorted by field name, these fields will be stored in adjacent records. There is a risk that the wrong fields will be compared, which would result in changes being reported where no change has occurred. The solution is to sort the records by sequence number as well as by field name.

3.2 The Field Properties

Each field and answer category should be sufficiently well described to allow for a detailed comparison. The properties we have selected include:

1) Field Name
2) Field Sequence Number
3) Field Path
4) Field Text
5) Field Type
6) Field Attributes
7) String Length (for String type fields)
8) Lower Bound (for Numeric type fields)
9) Upper Bound (for Numeric type fields)
10) Number of Codes (for Enumerated and Set type fields)
11) Number of Choices (for Set type fields)
12) Answer Index (for Enumerated and Set type fields)
13) Answer Code (for Enumerated and Set type fields)
14) Answer Text (for Enumerated and Set type fields)

Some of the above properties may need further explanation. The *answer index* can differ from the actual answer code. The Blaise programmer may have defined the answer code, but the answer index is always incremented by one. The *field path* gives the full field name, including the names of enclosing block fields, separated by dots. The *field type* is defined as Enumerated, Set, Integer, Real, String, Date, Time, Open or Class.

Not all the properties apply to all the fields. For example, fields of the types Date, Time and Open can only have meaningful values for the properties one to six. Field properties that don't apply are generally set to zero or one or a null string.

There are some other field properties that are worth considering. It may be useful to include the field description text (or field label) for data models that consistently use it. The field description gives the best indication of the meaning of the field.

4. Notes on Implementation

The Cameleon script allows the user to specify the level of the output. As mentioned above, one issue relating to the level of output is whether to record all the fields or just one instance of each field. The default setting is to select one instance, but the user can override this by telling the script not to suppress the repetition of arrays. If the array repetition is suppressed, the user can still select whether to output the first or the last element of the array. Selecting the last element means that the program will be able to detect changes to array sizes. This is possible because the field path property then records the highest index number of all the array elements.

It may cause a serious problem for the comparison of the Cameleon output if several fields share the same name. Sorting the records by sequence number as well as by field name only works if the two data models have exactly the same number of these fields. Some data models may have special identifier fields in repeated blocks that stores the situation, person, instance, etc, that the block relates to. If each of the blocks has a unique name (instead of being declared as an array) and the number of these blocks differs between the data models, the comparison won't work. Therefore, an additional parameter to Cameleon allows the user to specify any field name that should be excluded from the output files.

Comparing the two Cameleon output files is a routine data processing task. Pairs of records are compared. If a record is missing in one of the files, it is either because of a deletion or an insertion. If the pair has exactly the same contents, it is ignored. If they differ, the program has to establish in what way. The current program applies a priority order of field properties, to avoid over-reporting of fields that have changed in more ways than one. This could be amended so that the user can select whether to report only the most important change for a field or all changes for a field.

Most of the field properties are compared literally. It is only the text properties that require some special handling. We have designed the current program to take a very simple approach. Only letters, digits and brackets are included in the comparison and all the letters are converted to upper case. This means, for example, that adding or deleting a comma will not be reported as a text change whereas renaming a text substitution variable will be reported.

5 The Report Output

There are eleven possible report files, five of which are for printing and six of which are prepared for further processing. The reason for the latter is that the principal end user wants to load the files into Excel spreadsheets.

The non-printed files contain the Cameleon record from the old data model and, if applicable, the Cameleon record from the new data model. As all the field properties are comma separated, the records can easily be imported into Excel. These six files contain deleted fields, inserted fields, deleted answer categories, inserted answer categories, changed fields in field name and changed fields in sequence order.

The printed files only document the changed fields. Two files record all the changes, one in field name order and one in field sequence order. The contents of the former is then further split into a file recording text changes, a file recording field type changes and a file recording changes to the block or array structure. The latter distinction is possible because the field path is included as one of the field properties.

6. Different Versions

The system for documenting change between successive data models was originally developed in a DOS environment. The user starts by running the Cameleon script on both data models, copies the output files to a common directory and runs a DOS batch file. The batch file takes two parameters: the name of the old data model and the name of the new data model. At the core of the batch run is a QuickBasic program that compares the two files and generates the output files.

The system has subsequently been updated to run in Windows using only Blaise 4 Windows tools. A Maniplus shell, that collects information from the user and runs the appropriate Cameleon and Manipula routines, controls the processing. A Manipula setup converts the Cameleon output into Blaise data files. Additional Manipula setups compare the two data files and generate the report files. All the records that differ between the two data models are stored in a single Blaise data file. In the Windows version the user can select either or both of the report file formats.

The Windows design is clearly preferable to the DOS version. The user is able to select the data models, set the parameters and specify the report format in one single environment. There is, however, one small difficulty: the inability to store and process long strings in Blaise. It is not possible to store the field text in a single data structure, as a field, an auxfield or a local. The only way around this is to store the field text as an array of strings.

7. Conclusion

The documentation of change between successive data models is already being used with great success for the Family Resources Survey. It has also proved useful in other surveys. It could provide even more benefits if a future Cameleon has access to more information from the RULES sections. For example, knowledge about the number of edits in each block would make it possible to quickly find out why two consecutive data models are incompatible.