MultiRelaX with BCP as a replacement for Cameleon

Marien Lina (Statistics Netherlands) & Richard Frey (Westat, USA)

1. Introduction

This paper examines whether the Blaise Component Pack (BCP) can be used as a replacement for Cameleon when generating Datamodel descriptions. Cameleon is available in the Blaise system to retrieve meta data from the Blaise meta file and translate them into meta-descriptions and formats for ASCII and ASCII-relational output. Cameleon allows you to control your output when writing set-up files for SAS, SPSS, oracle, paradox, and for many other software applications. Blaise Data can be exported according to these formats with Manipula.

The recently developed Blaise Component Pack provides an opportunity to address data and meta information together in one program. The BCP enables you to create applications that prepare meta descriptions for software packages and at the same time write data in the related format. This paper discusses the outcome of an experiment in which a BCP application developed in VB writes, like Cameleon, related meta information. The meta descriptions for this application were designed to generate a language neutral output. The next step of this experiment is to write data, like Manipula, in the specified format of the exported Datamodel descriptions. The application constructed in this experiment has been named: MULTI RELational eXport tool, in short: MultiRelaX.

2. Exporting Datamodel Descriptions with Multi RelaX

The application is designed to read the Blaise meta information and the Blaise data base to create ASCII Relational output files. The output files are based on the block structure of the Datamodel where the information and data for each block are written to separate output files. Any fields in an embedded block are treated as being part of the parent block, and embedded parent blocks are treated as being part of their parent block.

The application was developed in Visual Basic, using the Blaise Component Pack to read meta information from the Blaise Meta file, generate language neutral descriptions and export Blaise data based on the description.

Exporting data in other formats requires more flexible options for writing meta information suitable to read the data in other packages. Obviously there are many ways to export data, asking for different meta descriptions.

The first step in the process is the creation of a general ***Meta Table.*** Three table types are part of this meta table: the ***Field Definitions Table***, the ***Long String Table*** and the ***Type Definition Table***. These tables identify the elements of the descriptions depending on the selected options.

The second step in the process is the selection of the layout for the exported meta data. Default options of the application can be changed using: ***File Options***, ***Field Options***, ***Long String Options***, and ***Type Options***. These options are partially related to the treatment of data, such as data selection or the truncated values of certain string fields. It is also related to layout options for the external meta (such as used keywords for types). The selection of options for the meta fields affects the export format of the data. The general advice is to use the same options when exporting meta and later, when exporting data. The final step, ***Data Export,*** is still

to be developed. For the moment the application deals with creating the meta files. The idea is to add facilities to define more tables and to control the writing of the data in specific formats.

3. The Meta Table

Producing the meta table is the first step to facilitate the organization of the essential parts of meta information into a suitable format. This meta table can be used as a working table and contains parts of the BMI file that are required to produce meta descriptions for other packages. Using the Meta Table as a work table it is possible to change the layout without having to read the Blaise meta files over and over again.

3.1. The Field Definitions Table:

The Basic Field Table keeps track of relevant field definitions within a block, including the order in which field values are to be exported. For each field in the block a selected part of the meta data is listed. For the export table, required data elements can be included or excluded by selecting options in a "meta table options" dialog. Following is a complete list of the data in the work table:

- The Datamodel name
- The block name

and for every field:

- The field name
- The field type
- The type name
- The Array or Set indicator of the type
- The field position
- The field length
- The question text
- The question description
- The defined export value for DONTKNOW scores
- The defined export value for REFUSAL scores

The ***field name*** is copied from the Blaise meta. For fields in non-embedded blocks, both the Datamodel name and the hierarchical block name can optionally be included in the field name. For fields in embedded blocks, the field name includes all information of the hierarchical position of the field within the embedded block structure. It includes all parent block names until the first encountered non-embedded parent block.

The ***field type*** is one of the following basic Blaise types: integer, real, enumerated, string, open, datetype, timetype, classification or external. The displayed export keyword for the types can be changed in the meta options. The field length can also be displayed in the field type.

The ***type name*** is an identifier for the field type. By default this is the user defined type name in the Blaise meta. If the field does not have a named type then the field name will be used as type name. The ***field type*** and ***type name*** are single types. Array and Set indicators are stored separately.

The ***Array or Set indicators*** indicate the field is an Array or Set, and keep track of the number of array cells. If a field is an array or a set, this will be available in the field definitions table of the work table. This information can eventually be used,

for example when selecting an export format in ARRAY notation, which has not yet been implemented.

The ***field position*** and the ***field length*** hold information about the precise position of the exported field.

The ***question text*** and the ***description text*** are field texts as defined in the Blaise meta.

The ***defined export value*** for **"*dontknow*"** or **"*refusal*"** values in Blaise are the scores that automatically would be written when exporting Blaise data to ASCII format with Manipula.

3.2. Long String Table

Apart from the field definitions table, there is a ***Long String Table.*** The reason for using this sub tables is related to the length of strings. Strings and memo fields may be large and have the potential to cause the export data file to exceed the size limits of some database systems. To avoid this problem the user has been given the option to exclude memo fields and strings or specify a maximum field length if the field is to be included in the export file. The Long String Table lists all fields that are excluded from the export data caused by oversized string length. The Data Output table for these fields will include records with the field name, the form key and the string value.

3.3. Type Definition Table

The ***Type Definition Table*** holds the Type meta descriptions of all found types in the Datamodel. This table is a flat list of all separately defined field types in the entire data model. At this moment there is only one type definition table.

If a type is repeated for an array or a set, the type definitions only include the first element of the type. Including the others is not necessary as the other elements are of the same type. Nevertheless, the type definition table can include similar types, if they are defined more than once in the Blaise Datamodel.

By default the type meta descriptions take the name of the type defined in a TYPE section. The TYPE section can be on the main level in the Datamodel hierarchy, but can also be defined inside a block. If the type is defined in a block, then the type name of the parent block (a block type) is included in the type name. These type names are copied from the Blaise meta descriptions.

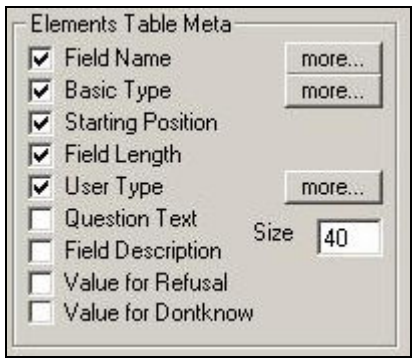
In Blaise you can define types directly after the field name and not in a TYPE, BLOCK or TABLE section. These types are not named in Blaise meta. For export meta it may be convenient, in some cases, to give these types a name. Therefore, the decision was to use the field name as the type identifier when there is no user defined type name. However, in Blaise, fields in different blocks may have the same name but a different type definition. Therefore, the preceding parent block name is included in the field name. This ensures that all types in the *Type Definitions Table* have a unique name.

4. Options

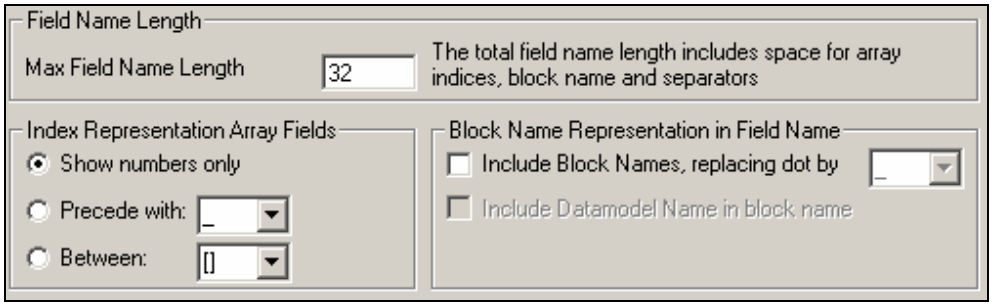
The application offers additional options for the formatting of the Meta description output and the substitution of some data attributes. Note that all the displayed dialogs presented in this paper are subject to change.

4.1. Field Options

The field options are the elements to be included in the field definition table. One can select whether to include the described options of the meta table in the field definitions table, for example, including the field name, basic type and the question text. The ”Size” box enables you to truncate the question texts and descriptions in your field definition table, in this case to 40 positions. All options can be included or excluded individually. In the current prototype this can be done in the following dialog:

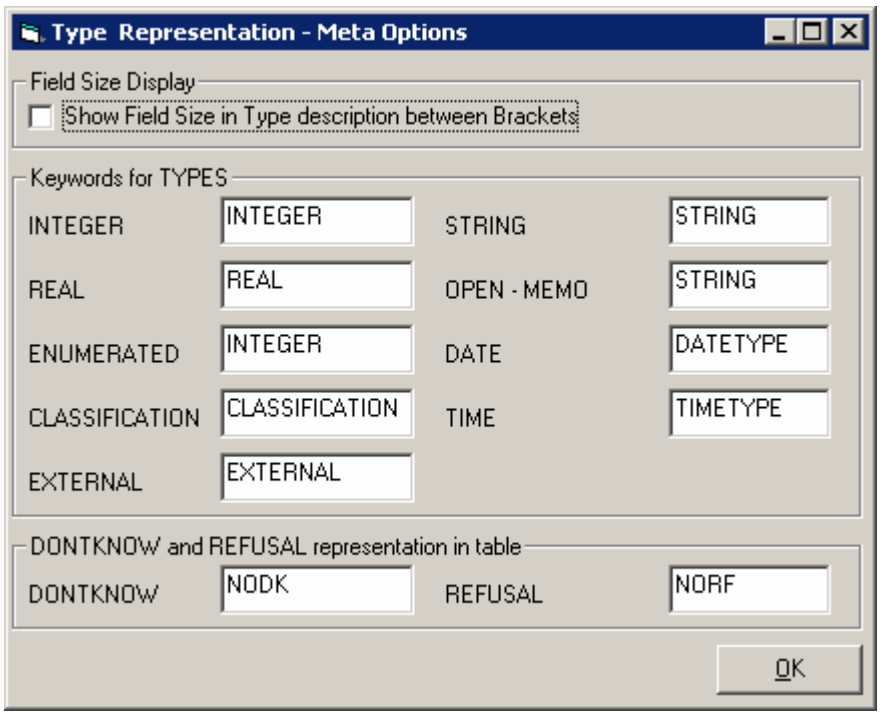


For some elements there are “more…” buttons. If you click the “more…” button next to the Field Name then the following items can be specified:



Here you can specify additional characteristics for displaying the field name, whether a block name should be included in the field name, the maximum field name length and how to array numbers should be treated and displayed.

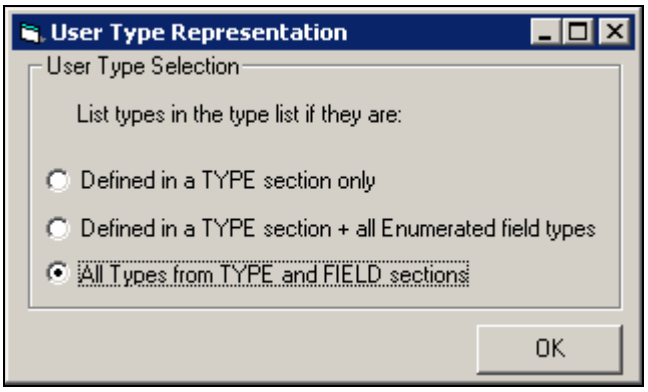
For the basic type there is also a “more…” options button. After selecting this button a dialog appears in which you can define keywords for the field definition table. The words will also be used for the type definition table. This is to meet requirements for data descriptions in different languages, for example, you may want to have a data description in which the Blaise type INTEGER is represented as LONG, and STRING as TEXT.



In the above dialoge you can specify the keywords for type specifications to be used in the field definition table. The default type names are displayed in the dialog and for each type you can change the type name you want to use in your tables. The dialog enables you to define other keywords for a type. For example, you could display the word FLOAT in stead of REAL in the definition table. These keywords will also be used for the type table.

Another option of this dialog is the specification to display the size of the field in the type. For example, if you have a string variable of 20 characters, you can select whether you want to see STRING (the default) or STRING[20] in the table.

The third “more…” options dialog is related to the user types.



By default a type list is made for all types that are defined in the data model. For a large data model one may include the types declared in a type section, striking out all types declared in the field section. In that case you will miss value labels of enumerated types declared in a field section. Therefore it has been made possible to select the user defined types and add enumerated types.

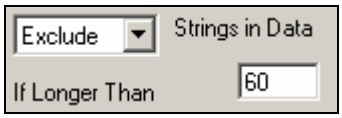
4.2. Additional options for the field definition table.

In the field descriptions section of the ***file options*** dialog you can select a separator and delimiter for the field definitions table. You can also indicate whether to print table headers and whether to truncate question texts and descriptions.

A header can be included for every block in the field definition table. The following dialog gives you also the possiility to change the default separators and string delimitors in the meta table.

4.3. Long String Options.

The main choice here is whether a string that exceeds a certain length should be excluded from the export data. The user can specify a maximum string length for strings to be excluded from the export data. The string will be written to a separate output table.



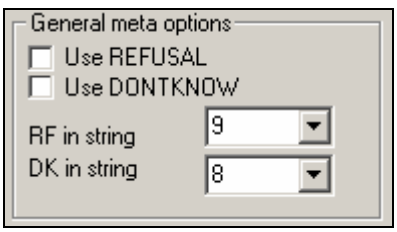
The meta definitions for the long strings excluded from the field definitions for the export tables is just a list of fields that are to be excluded. The options for the elements displayed for these fields is not very relevant as the required meta information can be included in the long string data.

4.4. Type Options.

As part of the field definition, this options also affect the representation of types in the type definition tables. This includes the choice of keywords for the basic types. It also includes the type selection of user types. In addition, it is possible to change the used separator and delimiter in the type table. This is done in the illustrated entry fields:

4.5. *Dontknow* and *Refusal* recognition in the export.

Other options in the main dialog enable the user to ignore dontknow and refusal scores, or to include them in the export file. This can be done in this dialog.



If the values are exported, these standard values may be taken. For numerical variables the value of a dontknow and refusal is numerical. For strings you may wish to specify specific characters as a dontknow value or refusal value in exported data, For example, one could change “9” into “#”, and that character could be exported if a string field has a Refusal score.

4.6. File Options.

One part of the *file options* enables the user to identify the locations of the input files (BMI and BDB) and the output folders for the export meta and data files.

After pressing on of the "…" button a dialog appears for specifying input files and output folders

5. Export.

5.1. Field definition output.

Once the options are selected and a BMI file has been selected, you can preview the generated meta tables. All options can be changed and previewed before exporting. A part of such a field definitions preview could look like this:

```
HaveNanny;INTEGER; 512;1;typYesNo;
HaveJob;INTEGER; 573;1;typYesNo;
PreviousJob;INTEGER; 574;1;typYesNo;
ThankYou;STRING; 575;1;ThankYou;
Nanny_NanName;STRING; 576;15;blkNanny_NanName;
Nanny_NanCountry;STRING; 591;15;blkNanny_NanCountry;
Paparazzi;INTEGER; 2;1;blkJob.zp;
CoName;STRING; 3;20;blkJob_CoName;
StartJob;DATETYPE; 23;8;blkJob_StartJob;
Region;INTEGER; 31;1;typRegion;
Paparazzi;INTEGER; 2;1;blkJob.zp;
CoName;STRING; 3;20;blkJob_CoName;
StartJob;DATETYPE; 23;8;blkJob_StartJob;
Region;INTEGER; 31;1;typRegion;
```

In this case the table displays a field name, field type, start position, the field size and a type name. After changing some options, a preview might be changed completely. One could instruct to make a variable lists (including name, type and size indicators) first. Then one could define another table: a list of variable names and labels (questionnaire text). For example, the following table, using the same fields as the list above, contains a very short field definition table with only field name and field type including field length:

```
HaveNanny;INT[1];
HaveJob;INT[1];
PreviousJob;INT[1];
ThankYou;TEXT[1];
Nanny_NanName;TEXT[15];
Nanny_NanCountry;TEXT[15];
Paparazzi;INT[1];
CoName;TEXT[20];
StartJob;DATETYPE[8];
Region;INT[1];
Paparazzi;INT[1];
CoName;TEXT[20];
StartJob;DATETYPE[8];
Region;INT[1];
```

If the result is what you want you can save the table in a file. After that you can create a second table with field name and field text only. By changing some options you would have the following table:

```
HaveNanny;"Do you have a nanny?";
HaveJob;"Do you have a job?";
PreviousJob;"Did you have any previous jobs?";
ThankYou;"Thank you for paticipating";
Nanny_NanName;"What is your nanny's name?";
Nanny_NanCountry;"What country is your nanny from?";
Paparazzi;"";
CoName;"What is the company name?";
StartJob;"When did you start working at ^CoName?";
Region;"Which region is ^CoName located?";
Paparazzi;"";
CoName;"What is the company name?";
StartJob;"When did you start working at ^CoName?";
Region;"Which region is ^CoName located?";
```

The separator ";" in the table can be changed into a space or any other character or string. These previews can be saved one by one.

5.2. Meta information for excluded long string fields

The long string data can be stored in a separate export file. For every excluded field the file will contain the record key, the block name, the field name and the string value. This is a point of concern when writing the data and it does not require a separate meta information table. In fact this meta would just be a list of the fields with long strings that are to be excluded form the export meta. This output can eventually be skipped as the required meta information will be included in the export data.

5.3. Type Definition output

Here is an example of a type definition table, the layout is still being developed. Each type is displayed over several lines. The included block name *blkNanny* is the name of the block in which the type *NanCountry* has been declared.

```
blkNanny_NanCountry
STRING[15];

blkJob.zp
 1;Cocina;"";
 2;Mamino;"";
 3;Paponi;"";

blkJob_CoName
STRING[20];

blkJob_StartJob
DATETYPE;
 99999998;refusal
 99999999;dontknow
```

For the type definition table, different layouts will be available, for example, using different separators, and possibly options to include or exclude value labels and descriptions. Obviously, useful information can be read from the Blaise Meta, and it would be a small step to adapt the layout to produce value labels and type declarations for SAS and SPSS.

Also here the idea is to preview the tables until the desired format is created. Then the table can be exported and the status of the selected options can be saved in a table model. At the moment this paper has been written, the prototype is still being developed.

5.4. Exporting Data with Multi RelaX

The current prototype does not support data export and only produces meta definitions. This is the next step furthering the development of the current prototype.

6. Conclusion

The experiment showed that Meta information can be retrieved from Blaise Meta information files using the Blaise Component Pack, and it can be organized like Cameleon to produce meta information. Similarly BCP can be used to read Blaise data and export it in the same format as analyzed from the BMI files. The application shows that it is technically possible to make this a Cameleon replacement application using Blaise Component Pack.

Session 4.1: Connecting