Automatic Customization of Datasets Using Cameleon

Barbara S. Bibb & R.. Suresh, RTI International

1. Overview

Cameleon is the utility that extracts information from Blaise databases and data models. The default settings and options of Cameleon are usually sufficient for simple questionnaires in Blaise. For complex questionnaires which use nested blocks or other powerful features within Blaise, the default output of Cameleon would require additional effort on the part of the programmer to provide mapping of the variables to the questions or recoding of variables for ease of analyses. But there are additional features and constructs in Cameleon that allow information within the Blaise data model to be queried and manipulated so as to generate code in languages like SAS. These generated programs can provide the mappings or recode variables as desired. This paper provides the details for two such programs.

2. Converting SET type responses to toggle variables

Blaise SET type responses are stored in arrays. Each answer category selected is stored in an array element in the order it is collected. This method of data storage preserves both the response given and the order in which it was entered or chosen. However, during analysis, to know whether a response was chosen at all, each array element must be checked for the particular response of interest. It is easier for the analysts if there is a toggle variable for each of the responses to the SET question. This Cameleon procedure generates a SAS program to define and populate sets of toggle variables for each SET question.

There are Cameleon constructs that provide information about the SET variable properties. For example, NUMBEROFCHOICES gives the number of answer or response choices the respondent can choose for each SET variable. ANSWERCODE identifies the current response and NUMBEROFANSWERS provides the number of answer categories that exist for that particular item. By using other Cameleon constructs such as the TYPE and LOOP commands, the programmer can leverage the Cameleon tool to create and manipulate code.

Using this SET variable example, consider a new array of toggle variables. These variables will be generated and populated so that there is one variable for each possible answer category. The value of this variable is TRUE if the response was chosen and stored in any of the array elements for that item in the original Blaise database. This new variable is FALSE if the response was not chosen.

By default, the size of SET variable arrays in Blaise is determined by the number of answers the respondent can enter (NUMBEROFCHIOCES). Whenever the number of answers allowed is less than the NUMBEROFANSWERS or actual categories, adjustments must be made so that the correct number of toggle variables is created. That is, the newly produced code must generate more variables than exist in the original data model.

The Blaise software package includes a number of *.CIF (Cameleon Translator) files. These files are Cameleon code files that produce basic code needed to convert the Blaise data models to various software applications like SAS, SPSS, Oracle, etc. These very useful files can be enhanced to produce alternate or modified code that may be more useful under certain circumstances.

The example shown starts with the standard SAS.CIF. Code has been added to the SAS.CIF to generate a length statement for each of the newly created toggle variables. These variables are filled with a 1 for “category chosen”. Each toggle variable represents an answer choice. Additionally, both format and label statements are generated for these toggle variables. This

code will convert the Blaise ASCII data to a SAS dataset with the desired nonstandard manipulations and will provide SAS formats and labels for these newly created variables.

2.1 Step 1

To [PROCEDURE WriteAllTypes], code has been added to generate a value statement for the SAS PROC FORMAT. The loop will execute for each variable of type = SET.

```
[if type = SET then]
  [/]VALUE rcTE_[ENUMTYPENUMBER]F
    [:2]-1='Dontknow'
    [:2]-2='Refusal'
    [:2] 1='Category selected'
  [];
```

2.1.1 Result 1

The initial value statement shown below is output as a result of the original SAS.CIF. A newly generated value statement for the data that will be recoded is created by the addition of the new loop in STEP 1.

```
VALUE TE_1F
  1='blue'
  2='green'
  3='yellow'
  4='purple'
  5='white'
  6='black'
;

VALUE rcTE_1F
  -1='Dontknow'
  -2='Refusal'
   1='Category selected'
;
```

2.2 Step 2

The [PROCEDURE DEFINEALLTOGGLES] is created to generate a length statement for the new toggle variables.

```
[PROCEDURE DEFINEALLTOGGLES]
[BLOCKPROC]
  [FIELDSLOOP]
    [ARRAYLOOP]
      [IF TYPE = BLOCK THEN] [BLOCKCALL]
        [ELSE]        [IF TYPE = SET THEN]
            [FOR loopct := 1 TO NUMBEROFANSWERS DO]
             [:4]length[:2]rc[loopct][UNIQUENAME 8:]3.;
            [ENDDO]
          [ENDIF]

      [endif]
    [ENDARRAYLOOP]
  [ENDFIELDSLOOP]
```

```
[ENDBLOCKPROC]
[ENDPROCEDURE][*DefineAllToggles]
```

2.2.1 Result 2

The new toggle variables are named using the first array element name. Their number depends on the total number of answer categories (NUMBEROFANSWERS) available for the item.

```
  length  rc1set11   3;
  length  rc2set11   3;
  length  rc3set11   3;
  length  rc4set11   3;
  length  rc5set11   3;
  length  rc6set11   3;
```

2.3 Step 3

The [PROCEDURE WriteAllrecodes] sets the newly created toggle variables to 1 if the category corresponding to their value is in any position of the original Blaise variable array. It also sets the first toggle to "Don't Know" or "Refused" if that was the answer given by the respondent. The toggle variables do not retain any information about the order in which the categories were chosen. That information is available from the original SET variable array.

```
[PROCEDURE WriteAllrecodes]
[BLOCKPROC]
 [FIELDSLOOP]
  [ARRAYLOOP]
   [IF TYPE = BLOCK THEN] [BLOCKCALL]
   [ELSE]
   [IF TYPE = SET THEN][loopct:=1]
    [currsetfield:=UNIQUENAME]
    [SETLOOP][loopct2:=1]
     [answerloop]
     [thiscode:=ANSWERcode]
     [settxt:=''] [loopct3 :=1]
     [SETLOOP]
      [IF SETTXT = '' THEN]
      [settxt:= 'if ' + UNIQUENAME + ' = ' + STR(thiscode) + ' ']
      [ELSE][settxt:= SETTXT + ' or ' + UNIQUENAME + ' = ' + STR(thiscode) + '
']['ENDIF][loopct3 :=loopct3+1]
     [ENDSETLOOP]
      [if loopct = loopct2 then][settxt:= settxt + ' then']
      [:4][settxt 80:]
      [endif][loopct2:=loopct2+1]
     [endanswerloop]
      [:8]rc[loopct][currsetfield] = 1; [refcode:=thiscode+1][dkcode:=thiscode+2][if dkcode < 9
then][refcode:=8][dkcode:=9][elseif dkcode < 99
then][refcode:=98][dkcode:=99][else][refcode:=998][dkcode:=999][endif]
      [if loopct=1 then]
      [:4]else if [UNIQUENAME] = [DKCODE] then
      [:8]rc[loopct][currsetfield] = -1;
      [:4]else if [UNIQUENAME] = [refcode] then
      [:8]rc[loopct][currsetfield] = -2;
      [endif]
      [loopct:=loopct+1]
```

```
      [ENDSETLOOP]
      [topnum := NUMBEROFANSWERS]
      [while loopct < topnum do]
       [SETLOOP][loopct2:= 1]
       [answerloop]
       [thiscode:=loopct]
       [settxt:=''] [loopct3 :=1]
       [SETLOOP]
        [IF SETTXT = '' THEN]
        [settxt:= 'if ' + UNIQUENAME + ' = ' + STR(thiscode) + ' ']
        [ELSE][settxt:= SETTXT + ' or ' + UNIQUENAME + ' = ' + STR(thiscode) + '
']['ENDIF][loopct3 :=loopct3+1]
       [ENDSETLOOP]
        [if loopct2 = NUMBEROFANSWERS then][settxt:= settxt + ' then']
        [:4][settxt 80:]
        [endif][loopct2:=loopct2+1]
       [endanswerloop]
        [:8]rc[loopct][currsetfield] = 1;
        [loopct := loopct+1]

       [ENDSETLOOP]
      [enddo]
     [ENDIF]
   [ENDIF]
   [ENDARRAYLOOP]
  [ENDFIELDSLOOP]
[ENDBLOCKPROC]
[ENDPROCEDURE][*WriteAllrecodes]
```

2.3.1 Result 3

The following SAS code is produced.

```
if set11 = 1  or set12 = 1  or set13 = 1  or set14 = 1  or set15 = 1  or set16 = 1  then
       rc1set11 = 1;
   else if set11 = 9 then          /** set of ( 6 categories) variable**/
       rc1set11 = -1;
   else if set11 = 8 then
       rc1set11 = -2;
if set11 = 2  or set12 = 2  or set13 = 2  or set14 = 2  or set15 = 2  or set16 = 2  then
       rc2set11 = 1;
if set11 = 3  or set12 = 3  or set13 = 3  or set14 = 3  or set15 = 3  or set16 = 3  then
       rc3set11 = 1;
if set11 = 4  or set12 = 4  or set13 = 4  or set14 = 4  or set15 = 4  or set16 = 4  then
       rc4set11 = 1;
if set11 = 5  or set12 = 5  or set13 = 5  or set14 = 5  or set15 = 5  or set16 = 5  then
       rc5set11 = 1;
if set11 = 6  or set12 = 6  or set13 = 6  or set14 = 6  or set15 = 6  or set16 = 6  then
       rc6set11 = 1;

   if set21 = 1  or set22 = 1  or set23 = 1  then
       rc1set21 = 1;
   else if set21 = 9 then              /** set [3] of ( 6 categories) variable**/
       rc1set21 = -1;
   else if set21 = 8 then
       rc1set21 = -2;
   if set21 = 2  or set22 = 2  or set23 = 2  then
```

```
      rc2set21 = 1;
   if set21 = 3  or set22 = 3  or set23 = 3  then
      rc3set21 = 1;
   if set21 = 4  or set22 = 4  or set23 = 4  then
      rc4set21 = 1;
   if set21 = 5  or set22 = 5  or set23 = 5  then
      rc5set21 = 1;
   if set21 = 6  or set22 = 6  or set23 = 6  then
      rc6set21 = 1;
```

2.4 Step 4

The procedures ToggleWriteLabels and ToggleWriteFormats were added to the stream. They mimic the existing procedures WriteLabels and WriteFormats but address only "set" variables.

```
[PROCEDURE ToggleWriteLabels]
  [BLOCKPROC]
    [FIELDSLOOP]
      [ARRAYLOOP]
        [IF TYPE = BLOCK THEN] [BLOCKCALL]
        [ELSE]
          [if type = set then][catct:=1]
            [for loopct := 1 to numberofanswers do]
[if catct = 1 then][rangetxt:= '(1, blank,-1=REF,-2=DK)' ][else][rangetxt:='(1 or
blank)'][endif][cattxt:= ' cat ' + str(catct) + ' selected ' +rangetxt]
[:2]rc[LOOPCT][UNIQUENAME 8:] = '[COPY(FIELDLABEL, 1 ,
60)][cattxt][catct:=catct+1]'
            [enddo]
          [endif]
        [ENDIF]
      [ENDARRAYLOOP]
    [ENDFIELDSLOOP]
  [ENDBLOCKPROC]
[ENDPROCEDURE][*ToggleWriteLabels]

[PROCEDURE TOGGLEWriteFormats]
[BLOCKPROC]
  [FIELDSLOOP]
    [ARRAYLOOP]
      [IF TYPE = BLOCK THEN] [BLOCKCALL]
      [ELSEIF TYPE = ENUMERATED OR TYPE = SET THEN]
        [IF TYPE = SET THEN]
         [FOR LOOPCT := 1 TO NUMBEROFANSWERS DO]
         [:2]rc[LOOPCT][UNIQUENAME 8:]rcTE_[ENUMTYPENUMBER]F.
         [ENDDO]
        [ENDIF]
      [ENDIF]
    [ENDARRAYLOOP]
  [ENDFIELDSLOOP]
[ENDBLOCKPROC]
[ENDPROCEDURE][*TOGGLEWriteFormats]
```

2.4.1 Result 4

Example code produced using these modified procedures is:

```
LABEL
  rc1set11    = 'code as many as there are cat 1 selected (1, blank,-1=REF,-2=DK)'
  rc2set11    = 'code as many as there are cat 2 selected (1 or blank)'
  rc3set11    = 'code as many as there are cat 3 selected (1 or blank)'
  rc4set11    = 'code as many as there are cat 4 selected (1 or blank)'
  rc5set11    = 'code as many as there are cat 5 selected (1 or blank)'
  rc6set11    = 'code as many as there are cat 6 selected (1 or blank)'
  rc1set21    = 'code only up to 3 cat 1 selected (1, blank,-1=REF,-2=DK)'
  rc2set21    = 'code only up to 3 cat 2 selected (1 or blank)'
  rc3set21    = 'code only up to 3 cat 3 selected (1 or blank)'
  rc4set21    = 'code only up to 3 cat 4 selected (1 or blank)'
  rc5set21    = 'code only up to 3 cat 5 selected (1 or blank)'
  rc6set21    = 'code only up to 3 cat 6 selected (1 or blank)'
;

FORMAT
  rc1set11   rcTE_1F.
  rc2set11   rcTE_1F.
  rc3set11   rcTE_1F.
  rc4set11   rcTE_1F.
  rc5set11   rcTE_1F.
  rc6set11   rcTE_1F.
  rc1set21   rcTE_2F.
  rc2set21   rcTE_2F.
  rc3set21   rcTE_2F.
  rc4set21   rcTE_2F.
  rc5set21   rcTE_2F.
  rc6set21   rcTE_2F.
;
```

2.5 Summary, example 1

Code from the standard SAS.CIF was modified and enhanced to produce modified SAS code. This new code generates a new set of variables that are used to determine if a category was chosen by the respondent as an answer to a SET question. These variables can be checked when answer order is irrelevant and the analyst is only concerned with whether or not the category was chosen. Automated code to do this task causes all the SET variables to be converted to toggles and the programmer need not be concerned about locating individual SET variables in a large complex questionnaire

3. Generating RENAME statements for potentially confusing variables

Questionnaires in Blaise can use nested blocks, SET questions, and other features that allow looping over blocks of questions. The field names within Blaise use “dot notation” for fully qualified path names. But when the data are extracted into SAS or other similar databases, Blaise generates variable names that are simply consecutive numbers. This essentially renumbers the variables from the original questionnaire and hence, requires the analysts to do additional work in mapping from Blaise generated variable names to analytical variable names.

Consider the example above that uses “set” variables. At the beginning of the sequence, the set variables are defined by Blaise for the item SET1 and named SET11, SET12, SET13, …SET16. If the questionnaire contained another item or question, called SET11, further down in the instrument, Blaise would assign it the next available variable number because SET11 has all ready been assigned by the data model.

To avoid confusion, final datasets should have variable names that correspond to the original item or question numbers. To achieve this end, a Cameleon cif can be used to extract the variable attributes from the Blaise datamodel. These variable attributes can be read and manipulated by alternate software to generate new code which will do the desired manipulations. For this example, variable information is read and RENAME statements are generated in SAS. The variables are renamed so that the variable names of the final data set correspond more closely with variable names of the questionnaire.

3.1 Step 1

By modifying the WriteAllFields procedure in the standard SAS cif, a *.txt file is generated that describes the variables in the datamodel. This procedure is outputting the UNIQUENAME, FIELDNAME, FIELDPATH, variable type (string/numeric), postion information, and various counts such as NUMBEROFCHOICES and NUMBEROFANSWERS.

```
[PROCEDURE WriteAllFields]
[VAR htxt ctxt: STRING  arryct SETCT: REAL]
[BLOCKPROC]
  [FIELDSLOOP]
    [arryct:=0]
    [ARRAYLOOP]
      [arryct := arryct+1]
      [setct:=0]
      [SETLOOP]
        [setct:= setct+1]
        [IF TYPE = BLOCK THEN] [BLOCKCALL]
        [ELSE]
          [IF TYPE <> OPEN THEN]
            [htxt:= UNIQUENAME]
            [ctxt:= '  ']
            [IF TYPE = STRING OR
               TYPE = DATE OR
               TYPE = TIME OR
               TYPE = CLASSIFICATION THEN]
               [ctxt:= ' $']
            [ELSE]
              [ctxt:= '  ']
            [ENDIF]
            [:4][htxt 11:][:2][ctxt 4:][FIRSTPOSITION :PositionWidth][:2][LASTPOSITION
:PositionWidth][:4][fieldname 20:][:2][arryct 3:][:2][setct
3:][:2][NUMBEROFCHOICES][:2][NUMBEROFANSWERS][:2][fieldpath]
          [ENDIF]
        [ENDIF]
      [ENDSETLOOP]
    [ENDARRAYLOOP]
  [ENDFIELDSLOOP]
[ENDBLOCKPROC]
[ENDPROCEDURE][*WriteAllFields]
```

3.1.1 Result 1

Output from the WRITEALLFIELDS modified code is:

```
  set11        1  1   set1          1   1   6  6  set1[1]
  set12        2  2   set1          1   2   6  6  set1[2]
  set13        3  3   set1          1   3   6  6  set1[3]
```

set14	4	4	set1	1	4	6	6	set1[4]
set15	5	5	set1	1	5	6	6	set1[5]
set16	6	6	set1	1	6	6	6	set1[6]
set21	7	7	set2	1	1	3	6	set2[1]
set22	8	8	set2	1	2	3	6	set2[2]
set23	9	9	set2	1	3	3	6	set2[3]
single	10	10	single	1	1	1	6	single
set17	11	11	set11	1	1	1	2	set11
set18	12	12	set12	1	1	1	2	set12
set131	13	13	set13	1	1	1	6	set13[1]
set132	14	14	set13	2	1	1	6	set13[2]
set133	15	15	set13	3	1	1	6	set13[3]
set134	16	16	set13	4	1	1	6	set13[4]

3.2 Step 2

Read the above listing with SAS to generate RENAME statements. The UNIQUENAMES for the fieldname SET1 are set11 through set16. There is also a fieldname called SET11. So, when Blaise generates the UNIQUENAME for fieldname SET11, it assigns the name SET17. Similarly fieldname SET12 gets the name SET18. This can be very confusing for the analyst who needs to use the extracted datafile.

To be consistent with the original FIELDNAMES, the following RENAME was produced by manipulating the variable attribute listing using SAS. Each ”set” variable was renamed with an underscore and an s (_s). Each array variable was renamed using an underscore and an a (_a). The variables that were assigned deceiving UNIQUENAMES were renamed back to the original FIELDNAME. Comment code was added to display the fully qualified FIELDPATH clarifying the reason for the RENAME of the variable. Where FIELDNAME = UNIQUNAME, the rename is unnecessary and no code is generated.

3.2.1 Result 2

```
RENAME
      set11  = set1_s1 /**set1[1] **/
      set12  = set1_s2 /**set1[2] **/
      set13  = set1_s3 /**set1[3] **/
      set14  = set1_s4 /**set1[4] **/
      set15  = set1_s5 /**set1[5] **/
      set16  = set1_s6 /**set1[6] **/
      set17  = set11 /**set11 **/
      set18  = set12 /**set12 **/
      set131  = set13_a1 /**set13[1] **/
      set132  = set13_a2 /**set13[2] **/
      set133  = set13_a3 /**set13[3] **/
      set134  = set13_a4 /**set13[4] **/
      set21  = set2_s1 /**set2[1] **/
      set22  = set2_s2 /**set2[2] **/
      set23  = set2_s3 /**set2[3] **/
;
```

3.3 Summary, example 2

For this example, the SAS.CIF was modified to extract a number of variable attributes from a Blaise data model. SAS was used directly to read the variable attributes and generate SAS statements that further modify the SAS dataset. Here, a RENAME statement was generated to clarify the variable names and make them correspond more closely with the original Blaise

fieldnames. A further extension of this technique might be to produce a list of variables to drop from a dataset with a DROP statement.

4. Conclusions

Use of a combination of Cameleon tools and other software can generate code to map or recode variables to make the final dataset easier to use. Extracting data may require the mapping of questions and variables or specific types of recodes. Furthermore, variables can be created and the attributes can be changed or adjusted to satisfy specific requirements.
Automating these procedures is critical because

- ✓ the underlying questionnaire might change during development and production
- ✓ the questionnaire might have multiple SET type questions and/or nested blocks that are time consuming to identify manually
- ✓ mapping these fieldnames to analytical names manually is error-prone and these errors will be hard if not impossible to detect
- ✓ generating the toggles can be tedious and again, is error prone

The major benefit to automating these procedures is that the data is treated consistently and with fewer errors, if any. The savings in time and cost can be substantial and the end product will be robust and accurate.

5. References

Blaise for Windows 4.1 Cameleon Reference

6. Acknowledgements

The authors acknowledge input from various staff with the Research Computing Division of RTI International. The additional ideas of these people helped develop the concepts discussed in this document. The people involved are Lilia Filippenko, Gilbert Rodriguez, and Rita Thissen.