Using metadata in Manipula and Maniplus

G J Boris Allan, Richard Frey, Jim O'Reilly
Westat
International Blaise Users Conference, April 2012, London, UK

***Summary*:** *Traditionally in Blaise Manipula and Maniplus have been used to examine data and Cameleon to examine metadata. We discuss here several ways of integrating and extending these tools to address challenging operational issues related to capturing and reusing metadata, manipulating data across rounds of longitudinal studies, and identifying key parameters of partially completed cases.*

Introduction

Since metadata information can now be accessed from Manipula, it is possible to more fully use the power of this capability in Blaise. In the past we used one type of tool to look at data (Manipula/Maniplus) and another tool to look at metadata (Cameleon). As Cameleon is able to produce text files as output, one can use Cameleon to generate a Manipula program (which is text) to use the metadata extracted by Cameleon to examine a database. We illustrate this technique in three applications:

- Removing remarks in rollover programs
- Finding where instruments stop
- Automatic conversion of Blaise data to a SAS dataset

Example code for using Cameleon to create a file with Manipula source code is given in the appendix.

Removing remarks in rollover programs

Longitudinal surveys typically require that data from one or more previous rounds be carried over to the current round. When data are assigned from one field to another, any remarks associated with the field are also copied into the current round. This means that fields in the current round are already unintentionally supplied with comments from the prior round, and thus it is difficult to distinguish current remarks from previous round remarks. One solution is to set the remarks for all fields in the current database to empty as part of the data assignment process for the new round. An example of the Manipula procedure used to clear out the prior round remarks (called in the MANIPULATE section by `RemoveRemarksBlockProc('')`) is as follows:

```
PROCEDURE RemoveRemarksBlockProc
  PARAMETERS
    BlockName : STRING

  AUXFIELDS
    FieldName : STRING
    SectionName : STRING

  INSTRUCTIONS
    FieldName := Present.GETFIRSTFIELDNAME(BlockName)
    REPEAT
      IF (Present.GETFIELDINFO(FieldName, 'BASEFIELDTYPENAME') = 'BLOCK') THEN
        RemoveRemarksBlockProc(FieldName)
      ELSE
        Present.PUTREMARK(FieldName,'')
      ENDIF
      FieldName := Present.GETNEXTFIELDNAME(FieldName)
    UNTIL (FieldName = '')
ENDPROCEDURE
```

This code is set up to attempt to emulate Cameleon metadata loops. Note that `Present` is the current update/output file. The code performs the following actions:

- Identifies the first field name for the named block (the initial call specifies an empty block, taken to be the top level).
- Repeats the looping until there is no next field name (end of block).
- Calls the procedure with the new block name, if the field is a block.
- Makes the remark empty if the field is not a block.
- Gets the next field name.

We could also use a similar procedure to output remarks as shown in the Cameleon and Manipula combination shown in the appendix.

Finding where instruments stop

Studies, particularly those with long and complex instrumentation, often define a partially completed case status. For example, the interview has been mostly completed, but not totally completed. In this study the partially completed were given an operational status between 80 and 89. Since post-collection processes need to identify where exactly the data collection ended, the last question in the interview has to be located. The following program identifies this information for all partially completed cases in this study.

```
SETDESCRIPTION(ParentID)
IF (SUBSTRING(CAPIStatus,1,1) = '8') THEN
  InputFile1.CHECKRULES
  FieldName := InputFile1.GETNEXTROUTEFIELDNAME('', 'ASK')
  REPEAT
    OutputFile1.EndFieldName := FieldName
    FieldName := InputFile1.GETNEXTROUTEFIELDNAME(FieldName, 'ASK')
  UNTIL ((FieldNAME = '') OR  ((InputFile1.GETVALUE(FieldName) = '' ) AND
      (InputFile1.GETVALUE(InputFile1.GETNEXTROUTEFIELDNAME(FieldName, 'ASK')) = '' )))
  OutputFile1.WRITE
ENDIF
```

The steps are:

- Select only partial completes based on the operational definition.
- Check the rules for that case.
- Get the name of the next field on the route that is an ASK field, starting from the beginning.
- Repeat the loop for the next section of code.
- Store that the current field name in the output file buffer (as EndFieldName).
- Get the name of the next field on the route that is an ASK field.
- Repeat the code loop unless there are no more fields on the route with data.
- Write the contents of the output file buffer.

An example of the output is as follows:

```
XXXXXXXX|CFQ.CFQ310_F
XXXXXXXX|HEQ.HEQ570c_F
XXXXXXXX|NRQ.NRQ266_F
XXXXXXXX|Fsq.OriginsKeyParent[2].WhereBorn
XXXXXXXX|PPQ.PPQ270_F
XXXXXXXX|HEQ.HEQ400_F
XXXXXXXX|HEQ.HEQ210_F
XXXXXXXX|SSQ.SSQ010o_F
XXXXXXXX|CHQ.CHQ345n_F
XXXXXXXX|DWQ.DWQ060_F
XXXXXXXX|HEQ.HEQ393_F
XXXXXXXX|SSQ.SSQ010x_F
XXXXXXXX|Fsq.RelTable.Relations[3].RelationToChild
XXXXXXXX|Fsq.OriginsKeyParent[2].HowOld
XXXXXXXX|NRQ.Questions[2].NRQ122_F
XXXXXXXX|Fsq.OriginsKeyParent[2].WhereBorn
XXXXXXXX|SSQ.SSQ010x_F
XXXXXXXX|Fsq.EducationsKeyParent[2].FSQ221_F
XXXXXXXX|HEQ.HEQ370_F
```

This information is also being used to provide QC reports to field supervisors, so that they know how far field interviewers have progressed in the assigned interviews. Data from interviewed cases is captured whenever interviewers contact Home Office so that information is kept as current as possible about the last question completed during the interview. The program could be easily modified to capture other data about the interview as well.

Automatic conversion of Blaise data to a SAS dataset

Many studies have the need to convert a Blaise database with its datamodel into a SAS dataset with variable descriptions, formats, and a matching text input file. We have developed a WesBlaisetoSAS process consisting of two Maniplus programs: the first program asks for file names and folder locations, and the second (called by the first) uses a variable datamodel, and uses many temporary files during the creation of the SAS data definitions:
Users install WesBlaisetoSAS by running an executable file `InstallWesBlaiseSAS.exe`, accepting `C:\` as the default location to unzip files. The program creates a folder with a README.TXT file that lists the other files in the folder as follows:

- Manipula.exe (used by msu files to read and write Blaise information)
- RunSAS32.msu (used to collect details about files and folder locations)
- SAS32BlaiseDriver.msu (uses Blaise datamodel descriptions and Blaise data to produce a SAS program, which opens in SAS)
- WesBlaiseSAS.lnk (used to execute RunSAS32.msu to collect file and folder information and then produce the SAS program)
- westat.bmp (the Westat logo)

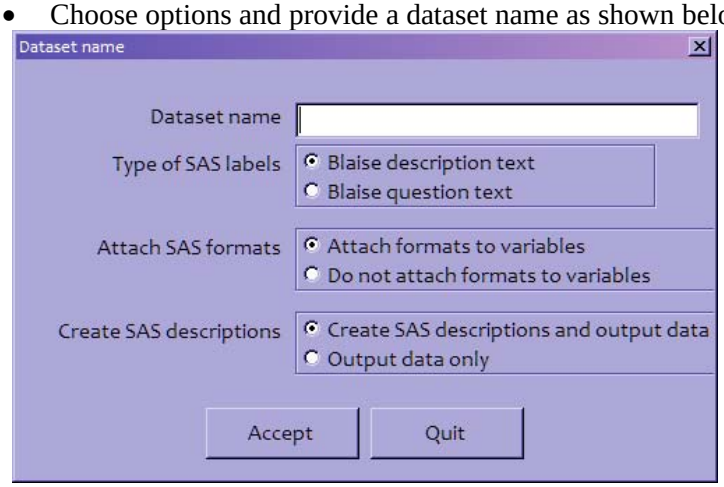
For users without a complete Blaise installation, the programs can be run by providing `Manipula.exe` and the prepared Manipula files. Such users start the program by use of the `WesBlaiseSAS.lnk` shortcut, or – if they have Blaise installed – start the prepared Manipula file `RunSAS32.msu`. The code for this file is as follows:

```
PROCESS CreateSASInstructionsFromBlaise
SETTINGS
  DESCRIPTION = 'WESTAT: Creating SAS dataset instructions for a Blaise database'

AUXFIELDS
  DatamodelLocation, DatabaseLocation, SASFolderFileLocation, SASFolderLocation : STRING
  SASFileName "What is the name of the SAS program?" / "Dataset name": STRING[50], EMPTY
  DescriptionOrQuestion "Do you want to use the Blaise description or the Blaise question text for
        labels?"
      / "Type of SAS labels" : (Description "Blaise description text", Question "Blaise question
        text")
  AttachFormats "Do you want to attach SAS formats to variables?"
      / "Attach SAS formats" : (Yes "Attach formats to variables", No "Do not attach formats to
        variables")
  DataOnly "Do you want create SAS descriptions?"
      / "Create SAS descriptions" : (Yes "Create SAS descriptions and output data", No "Output data
        only")

  AcceptName "Accept this file name" : (Yes, No)
  DoExit "Exit application" : (Yes, No)
  Rslt : INTEGER

DIALOGBOX TheFileName "Dataset name"
  FONTNAME = "CANDARA"
  FONTSIZE = 12
  SIZE = (500, 300)
  DEFAULTBUTTON = NO
  CONTROL SASFileName
    POSITION = (200,30)
    LABEL = Yes

  CONTROL DescriptionOrQuestion
    POSITION = (200,60)
    LABEL = Yes
```

```
  CONTROL AttachFormats
    POSITION = (200,120)
    LABEL = Yes

  CONTROL DataOnly
    POSITION = (200,180)
    LABEL = Yes

  BUTTON AcceptName
    CAPTION 'Accept'
    STORE = YES

  BUTTON DoExit
    CAPTION 'Quit'
    VALUE = Yes
    STORE = YES
MANIPULATE
  SETLOGO('westat.bmp')
  DatamodelLocation := '"' + SELECTFILE ('Select Blaise datamodel', '*.bmi', 'Blaise datamodels') +'"'
  DatabaseLocation := '"' + SELECTFILE ('Select Blaise database', '*.bdb', 'Blaise databases') +'"'
  SASFolderLocation := SELECTFOLDER('Select SAS data folder')
  IF (LEN(SASFolderLocation) > 0) THEN
    SASFolderLocation := '"' + SASFolderLocation + '\"'
  ELSE
    SASFolderLocation := '"'
  ENDIF
  TheFileName
  IF (DoExit <> Yes) THEN
    IF (POSITION(SASFileName + '==', '.SAS==') < 1) AND (LEN(SASFileName) > 0) THEN
      SASFileName := SASFileName + '.SAS'
    ENDIF
    SASFileName := '"' + SASFileName + '"'
    Rslt := CALL('SAS32BlaiseDriver.msu /KInputMeta=' + DatamodelLocation + ' /I'+ DatabaseLocation +
        ' /P' + SASFolderLocation+ ';' + SASFileName + ';' + STR(DescriptionOrQuestion) + ';' +
        STR(AttachFormats))
   ENDIF
```

The sequence of processing and information that users provide when running the program is:

- Select the Blaise datamodel
- Select the Blaise database
- Select the folder for the SAS program
- Choose options and provide a dataset name as shown below.



The options are:

- SAS labels
  - Use the Blaise descriptions for labels
  - Use Blaise question text for labels
- SAS formats
  - Attach formats to variables

    - Make a dataset where formats are not part of the variable description
- Creating SAS descriptions and outputting data
    - Both create SAS descriptions and create an output data text file
    - Create an output data text file (assumes the SAS descriptions are already available)

A second Manipula program runs using information collected by the first Manipula program. This second program uses the Blaise metadata to produce SAS descriptions describing the data, and uses Blaise data to produce a text file that matches the descriptions already produced. It starts:

```
PROCESS SASDataDescriptions
SETTINGS
  DESCRIPTION = 'WESTAT: Create SAS datamodel descriptions and extract data'
  ACCESS = SHARED
  CONNECT = YES

USES
  InputMeta (VAR)

  DATAMODEL UniqueMeta
    PRIMARY UniqueObject
    FIELDS
      UniqueObject : STRING[250]
      ObjectCode : STRING[8]
  ENDMODEL

  DATAMODEL OutputMeta
  FIELDS
    OutStr : STRING[1023]
  ENDMODEL

INPUTFILE BlaiseData : InputMeta('', BLAISE)
SETTINGS
  CHECKRULES = YES

TEMPORARYFILE UniqueFields : UniqueMeta
TEMPORARYFILE UniqueFormats : UniqueMeta
TEMPORARYFILE FormatDefsOutput : OutputMeta
TEMPORARYFILE VarPosnsOutput : OutputMeta
TEMPORARYFILE VarDescriptionsOutput : OutputMeta
TEMPORARYFILE VarFormatsOutput : OutputMeta


OUTPUTFILE SASOutput: OutputMeta ('', ASCII)
SETTINGS
  OPEN = NO
  TRAILINGSPACES = NO

OUTPUTFILE TextFile: InputMeta ('', ASCII)
SETTINGS
  OPEN = NO
```

Some of the intermediate code includes:

```
PROCEDURE ListUniqueObjects
  PARAMETERS
    IMPORT BlockName : STRING
  AUXFIELDS
    ObjectName, LocalName, UniqueObject, TypeName, BaseType, CategoryDescription, FieldDescription,
        StrSign : STRING
    UniqueID, Categories : INTEGER
  INSTRUCTIONS
    ObjectName := BlaiseData.GETFIRSTFIELDNAME(BlockName)
    StartPosn := EndPosn + 1
    REPEAT
      LocalName := BlaiseData.GETFIELDINFO(ObjectName, 'LOCALNAME')
      UniqueID := 1
      UniqueObject := ConvertToASCII127(LocalName,31 ,2, 1)
      SETDESCRIPTION('WESTAT: Create SAS datamodel descriptions -- ' + ObjectName)
```

```
        FieldDescription := REPLACE(FieldDescription, '''', '`')
        VarDescriptionsOutput.OutStr := '  ' + UniqueObject + ' = ''' + FieldDescription + ''''
        VarDescriptionsOutput.WRITE
        IF (UPPERCASE(BaseType) = 'ENUMERATION') THEN
          IF (UPPERCASE(TypeName) = '') THEN
            TypeName := REPLACE(ObjectName,'.','_')
          ENDIF
          TypeName := ConvertToASCII127(Typename + '_W', 31, 2, 2)
          UniqueFormats.GET(TypeName)
          IF NOT (UniqueFormats.RESULTOK) THEN
            UniqueFormats.UniqueObject := TypeName
            UniqueFormats.ObjectCode := 'W' + REPLACE(STR(FormatNum,6), ' ', '0') + 'W'
            UniqueFormats.WRITE
            FormatDefsOutput.OutStr := '  VALUE ' + UniqueFormats.ObjectCode
            FormatDefsOutPut.WRITE
            Formatnum := FormatNum + 1
            Categories := VAL(BlaiseData.GETFIELDINFO(ObjectName, 'CATEGORIES.COUNT'))
            FOR I := 1 TO Categories DO
              IF (BlaiseData.GETFIELDINFO(ObjectName, 'CATEGORIES[' + STR(I) + '].DEFINEDTEXT') <> '')
        THEN
                CategoryDescription := BlaiseData.GETFIELDINFO(ObjectName, 'CATEGORIES[' + STR(I) +
        '].DEFINEDTEXT')
              ELSE
                CategoryDescription := BlaiseData.GETFIELDINFO(ObjectName, 'CATEGORIES[' + STR(I) +
        '].NAME')
              ENDIF
              CategoryDescription := ConvertToASCII127(CategoryDescription, 31, 2, 2)
              FormatDefsOutput.OutStr := '    '+ BlaiseData.GETFIELDINFO(ObjectName, 'CATEGORIES[' +
        STR(I) + '].CODE') + ' = ' + '''' +  CategoryDescription + ''''
              FormatDefsOutput.WRITE
            ENDDO
            FormatDefsOutput.OutStr := '  ;'
            FormatDefsOutPut.WRITE
            FormatDefsOutput.OutStr := ''
            FormatDefsOutput.WRITE
          ENDIF
          VarFormatsOutput.OutStr := '  ' + UniqueObject  + '   ' + UniqueFormats.ObjectCode + '.'
          VarFormatsOutput.WRITE
        ENDIF
      ENDIF
      ObjectName := BlaiseData.GETNEXTFIELDNAME(ObjectName)
   UNTIL (ObjectName = '')
ENDPROCEDURE
```

After the completion of the program, the SAS screen is displayed with the appropriate output. Example outputs are shown below.

```
DATAMODEL Stopping
  TYPE
    tYesNo = (Yes, No)
    tReason = (Health (1) "@UHEALTH OR FUNCTIONING@U",
                  Other (2) "@UOTHER REASON@U")
  FIELDS
    One "Why do you want to stop?" / "Reason to stop" : tReason
    Two "Are you sure?" / "Confirm stopping": tYesNo
  RULES
     One
     Two
ENDMODEL
```

SAS program for the ‘Stopping’ datamodel:

```
TITLE 'StopInterview_Default';

PROC FORMAT;
  VALUE W000001W
    1 = 'HEALTH OR FUNCTIONING'
    2 = 'OTHER REASON'
  ;

  VALUE W000002W
    1 = 'Yes'
    2 = 'No'
```

```
  ;

RUN;

LIBNAME SAVEFILE "H:\MY DOCUMENTS\Blaise\";
DATA SAVEFILE.StopInterview_Default;

INFILE 'H:\MY DOCUMENTS\Blaise\StopInterview_Default.ASC' LRECL = 2;

INPUT
  One       1-1
  Two       2-2

;

LABEL

  One = 'Reason to stop'
  Two = 'Confirm stopping'
;

FORMAT
  One   W000001W.
  Two   W000002W.
;

RUN;
```

SAS program for question text labels:

```
TITLE 'StopInterview_QuestionText';

PROC FORMAT;
  VALUE W000001W
    1 = 'HEALTH OR FUNCTIONING'
    2 = 'OTHER REASON'
  ;

  VALUE W000002W
    1 = 'Yes'
    2 = 'No'
  ;

RUN;

LIBNAME SAVEFILE "H:\MY DOCUMENTS\Blaise\";
DATA SAVEFILE.StopInterview_QuestionText;

INFILE 'H:\MY DOCUMENTS\Blaise\StopInterview_QuestionText.ASC' LRECL = 2;

INPUT
  One       1-1
  Two       2-2

;

LABEL

  One = 'Why do you want to stop?'
  Two = 'Are you sure?'
;

FORMAT
  One   W000001W.
  Two   W000002W.
;

RUN;
```

SAS program for formats not attached to variables:

```
TITLE 'StopInterview_NoAttachedFormats';

PROC FORMAT;
  VALUE W000001W
    1 = 'HEALTH OR FUNCTIONING'
    2 = 'OTHER REASON'
  ;

  VALUE W000002W
    1 = 'Yes'
    2 = 'No'
  ;

RUN;

LIBNAME SAVEFILE "H:\MY DOCUMENTS\Blaise\";
DATA SAVEFILE.StopInterview_NoAttachedFormats;

INFILE 'H:\MY DOCUMENTS\Blaise\StopInterview_NoAttachedFormats.ASC' LRECL = 2;

INPUT
  One       1-1
  Two       2-2

;

LABEL

  One = 'Reason to stop'
  Two = 'Confirm stopping'
;

/*

FORMAT
  One   W000001W.
  Two   W000002W.
;

*/

RUN;
```

Appendix: Using Cameleon to create a file with Manipula source code

The idea is to extract all remarks, for every case in a database, using Manipula. This Cameleon script creates Manipula source code to do that task:

```
[* Writes Manipula code to extract all remarks for each case in a database]

[VAR
     PRIMARYFIELD : STRING,
     LabelText    : STRING,
     CharToken    : STRING,
     Quote        : STRING,
     I            : REAL
]
[MAXNAMELENGTH := 127]
[OUTFILE := DATAMODELNAME+'_Remarks.MAN']
[]USES
[]  InMeta '[DATAMODELNAME]'

[]  DATAMODEL PrintMeta
[]    FIELDS
[]      AuxOut : STRING[[1023]]
[]  ENDMODEL

[]INPUTFILE InFile : InMeta ('[DATAMODELNAME]', BLAISE)
[]OUTPUTFILE PrintFile : PrintMeta('[DATAMODELNAME]_Remarks.txt', PRINT)

[]MANIPULATE

[PROCEDURE RemoveQuotes]
```

```
  [Quote := '''']
  [LabelText := '']
  [FOR I :=1 TO LENGTH(FieldLabel) DO]
    [IF (I < 64) THEN]
      [CharToken := COPY(FieldLabel,I,1)]
      [IF (CharToken <> Quote) THEN]
        [LabelText := LabelText + CharToken]
      [ENDIF]
    [ENDIF]
  [ENDDO]
[ENDPROCEDURE]

[BLOCKPROC]
  [FIELDSLOOP]
    [IF PRIMKEYFIELD THEN]
      [PRIMARYFIELD := FIELDNAME]
    [ENDIF]
  [ENDFIELDSLOOP]
  [FIELDSLOOP]
    [ARRAYLOOP]
      [IF TYPE <> BLOCK THEN]
        [IF TYPE <> OPEN THEN]
          [SETLOOP]
            []  IF ([FIELDPATH] = REMARKED) THEN
            [RemoveQuotes]
            []    AuxOut := [PRIMARYFIELD] + '|[FIELDPATH]|[LabelText]|' +
        REPLACE(REPLACE([FIELDPATH].REMARK,CHAR(13),' '),CHAR(10),'')
            []    PrintFile.WRITE
            []  ENDIF
          [ENDSETLOOP]
        [ENDIF]
      [ELSE]
        [BLOCKCALL]
      [ENDIF]
    [ENDARRAYLOOP]
  [ENDFIELDSLOOP]
[ENDBLOCKPROC]
```

The following Manipula source is for datamodel HH01:

```
USES
  InMeta 'HH01'

  DATAMODEL PrintMeta
    FIELDS
      AuxOut : STRING[1023]
  ENDMODEL

INPUTFILE InFile : InMeta ('HH01', BLAISE)
OUTPUTFILE PrintFile : PrintMeta('HH01_Remarks.txt', PRINT)

MANIPULATE


  IF (Identifier = REMARKED) THEN
    AuxOut := Identifier + '|Identifier|What is the household identifier?|' +
        REPLACE(REPLACE(Identifier.REMARK,CHAR(13),' '),CHAR(10),'')
    PrintFile.WRITE
  ENDIF
  IF (ZipCode = REMARKED) THEN
    AuxOut := Identifier + '|ZipCode|What is the zip code for this household?|' +
        REPLACE(REPLACE(ZipCode.REMARK,CHAR(13),' '),CHAR(10),'')
    PrintFile.WRITE
  ENDIF
  IF (Household.Demographics[01].HHIndex = REMARKED) THEN
    AuxOut := Identifier + '|Household.Demographics[01].HHIndex|HH position|' +
        REPLACE(REPLACE(Household.Demographics[01].HHIndex.REMARK,CHAR(13),' '),CHAR(10),'')
    PrintFile.WRITE
  ENDIF
  IF (Household.Demographics[01].Name = REMARKED) THEN
    AuxOut := Identifier + '|Household.Demographics[01].Name|What is your name?|' +
        REPLACE(REPLACE(Household.Demographics[01].Name.REMARK,CHAR(13),' '),CHAR(10),'')
    PrintFile.WRITE
  ENDIF
```