PharmaSUG 2014 - Paper CC45

Self-fulfilling Macros Generating Macro Calls and Enabling Automation from a Specification

Y. Christina Song, Rho, Inc, Chapel Hill, NC

ABSTRACT

In most cases, macros facilitate repetitive SAS programming iterations. In clinical data cleaning tasks, like edit check programming, each query item has different query logic. However, the overall query report generation process is somewhat repetitive; it all includes the process of reading, subsetting, formatting, and printing data. Even with all of the macros, programmers still have to read specifications carefully and create macro calls accordingly. The overall process is time-consuming and labor intensive. To improve programming efficiency, a procedure is introduced to let SAS macros generate macro calls from the specification sheet and to do all of the SAS programing. This program takes advantage of the data-driven and dynamic features of SAS macros and dynamically reads specifications, tweaks the data, generates all macro calls, codes the texts between the macros, formats the data, and outputs the data into desired reports. This paper outlines the key elements and basic steps of the macros, and discusses how this strategy could be used to create other macros generating macro calls and enabling automatic operations. It may also be used for similar tasks that come with a specification sheet, such as generating some of standard analysis data sets.

INTRODUCTION

Often SAS programmers encounter tasks directed by specifications. Each item in the specification is different, but the overall processing is somewhat repetitive. For such tasks, we normally have macros to handle similar SAS programming iteration. For example, in clinical data cleaning, programming for each edit-check often uses the same macros for reading in data, formatting data, or creating desired reports. However, with all these macros, programmers still have to read the specifications carefully, and make macro calls accordingly. For processes like this, the programming task can be time-consuming, labor intensive, and repetitive in generating each query report.

This paper illustrates a method for allowing the powerful SAS to do all the tasks with the specification. The SAS macro codes will dynamically read specifications, tweak the data, generate all macro calls, code the text in between macros, and output data into the desired reports according to the specifications. This method takes advantage of data driven and dynamic features of the SAS macro. It uses common SAS procedures and requires a few key steps. Overall it markedly improves programming efficiency markedly with the following advantages:

1. It saves a substantial amount of time by eliminating the need to read specifications and program for each item and for each report.

2. Human error is minimized through such automation. Debugging is made easier by looking only at the specification, since all of the processing steps, such as reading, formatting, and printing data, are handled by the same macros.

3. It may be possible to update specifications, batch run the program, and output the report instantly without involving a programmer, and thus, speed up the review process.

4. This programming strategy also can be used in creating other automatic macros with specifications for similar tasks, perhaps some of the standard analysis datasets.

BASIC ELEMENTS

Edit check specification spread sheet if needed

Add program code column in the specification:
V1 < V2; or C1>' '.

SAS data step

Read the specification sheet into a SAS dataset:

check_name	str1	v1	str1varlist	str2	v2	str2varlist	str3	v3	str3varlist
LFTA-013	LFTA	GGTVAL1		DEMO	DOBDT	SEX	INEX	CONDATE	
LFTA-016	LFTA	TBVAL1	TBVAL2						
message1									
GGT value is not within the expected range of 4.4 - 125 when the Units are U/L, the subject is Male									
Total Bilirubin value is not within the expected range of 0.12 - 3.25 when the Units are g/ld.									

PROC SQL

Convert specification data to the required macro calls, e.g.:

```
%getmydata (l_num=013, v_num=1, str=LFTA, v=GGTVAL1, list1=GGTDAT, list2= , list3=…);
%Excel_To_X_check (l_name=LFTA , l_num=016, query_logic=( input (compress(v1, '<>'),
8.)>3.25 or input(compress(v1, '<>'), 8.)<0.12) and c1list2^='µmol/L' , by_visit= ,
message1="Total Bilirubin value is not within the expected range of 0.12 - 3.25 when
the Units are mg/dL. Please confirm or correct. ", message2=" ", str1=LFTA ,
v1=TBVAL1, str2= , v2= , str3= , v3= );
```

Macro calls working with existing macros

To read data from correct location, dataset, and/or with certain values;
To merge and subset data according to query logic;
To convert all variables to generic names: V1, V1list1-V1list10, etc.;
To convert all format of DATETIME to DATEPART etc.;
To format data with original names and string data for printing the report.

Print final listing reports according to the specification

Note: The program code column in the specification sheet is the part of the "where" or "if" statement in the last data step to subset the data. It dictates the program with query logic. It could be specified by DM as: field 1 should > field 2. Most likely, it will be coded by a programmer. The code can be as simple as: "V1<V2"; or as difficult as: "((input (compress(v1, '<unk>', 'i'), 8.)>80) or (input (compress(v1, '<unk>','i', 8.)<2.8)) and (floor((intck('month', v2, v3)-(day(v3)<day(v2)))/12)>15) and sex^='Male'".

KEY STEPS

The planning for overall programing strategy needs to be well thought out. Here are the key steps:

1. Make the specifications SAS friendly. This procedure basically uses the DM standard specification template at Rho, with some minor modifications. For example, the original title row can have text like “List Name”. Then another title row is added using SAS code-like text, “List_Name”, as variable names.

2. Associate the program name with each report, a group of list items. From the specifications, the query lists could be divided into several output reports, each requiring its own SAS program. Each program name subsets the proper query lists for each report according to the specification.

3. Create the logic code column, i.e. ‘v1>v2’. Or translate the column in the specifications to valid SAS code, if done by non-programmers. For example, “scrndt should = vistdt” on the specification will be translated using tranwrd function (varname, "should =", "NE") inside the macro. So we can output any records where scrndt NOT = vistdt.

4. Rename all variables in a generic, sequential manner for correct data logic operation. The generic name, like datasets str1, str2, str3 and variables v1, v2, v3 etc., makes it possible to store the specification information as macro variables. It will also help to avoid confusion when comparing two variables with the same variable name from different datasets, e.g., STDT in AEXP and STDT in DOSE.

5. Turn the specification into SAS macro calls. After all the preparations, this is the most important section in this program: allowing SAS to create all macro calls from the specifications. Here is an example of creating one macro call.

   1). First to define a new variable, e.g., getmydata1, in a data step to store the macro call text:

   ```
   Getmydata1 =compbl("%nrstr(%getmydata)(l_num="||l_num||", v_num= 1, str="
   ||str1 ||", v="||v1||")") ;
   ```

   2). Then to turn such text into macro variable using proc sql:

   ```
   Select getmydata1 into : call_getmydata1 separated by ' ; ' from &prog._x;
   ```

   3). Lastly, to call the macro, simply code it at the right place in the program as:

   ```
   &call_getmydata1;
   ```

   When the program is submitted, this macro-call will be resolved according to the specification, as shown in the log:

   ```
   %getmydata (l_num=001 , num= 1 , str=ELIG,   v=SCRDT ) ;
   ```

   In a similar manner, all macro calls in the program can be generated.

6. Make the in-between-macros code and finalize the program. First, program all macros that are needed. Then put the macro-calls and any in-between code in the correct sequences. The layout of the entire program is outlined in the previous section of “Basic Elements”.

APPLICATION

To run the program for each report, simply call the macro like this:

```
%EditChk_Noclip (   STUDY         =iWT0895,
                    PGMNAME       =List_G ,
                    PGMPATH       =&root \Programs\Listings_Programs,
                    OUTPUT        =&OUTDIR ,
                    Exl_in        =P:\Edit checks\Checks_DRAFT_v1.0_2013.xlsx,
                    DataLib       =&root\StudyData\Master,
                    DataPostFix   =MSTR,
                    Visit         =phase,
                    Site          =sitename,
                    Trimsite      =substrn(compress(sitename), 1, 3),
                    Subprint      =TESTING);
```

The macro call could pick the correct query lists for the report from its program name. Then it will choose the correct datasets and correct variable names and make the correct logic operations according the specification turned macro calls. At the end, it will format and print the query list as a report. After setting up the read in and output locations, as well as other basic project-related information, the only change needed for each query report is to change its program name.

This program, in theory, should be able to handle most edit check query requests. However, certain manual alteration will be required to improve the report. To provide almost endless flexibility, two final macro calls, %*FormatAll* and %*PrintAll*, are put outside the main macro as %*EditChk_Modif*. This will allow modification of data before formatting, and modification of output before printing:

```
*Exactly the same as noclip version;
      %EditChk_Modif (STUDY = …, subprint =… );
*To modify the data, add a data step here, then;
      %FormatAll;
*To modify the output style, add a data step here, then;
      %PrintAll;
```

Here is a sample without any manual modification:

Query: GGT value is not within the range of 0 - 57.5 when the unit is U/L and the subject was 1-15 years old when the lab was collected.

Subject	Site	visit/ Phase	Form1: Liver Tests/LIVR	Form2: Recipient Demographics / DEMO	Form3: Inclusion & Exclusion/INEX	DM Comment
10021	001	3012	GLTVAL(GGT Value U/L)=85	DOBDT(Date of Birth)=21JUL1999	CONDATE(Consent Date)=14AUG2012	
10043	004	3028	GLTVAL(GGT Value(U/L)=59	DOBDT(Date of Birth)=14JUL2001	CONDATE(Consent Date)=08AUG2012	

After %*EditChk_Modif* and before %*FormatAll*, insert a new data step to derive a new variable “Age”, and assign it to the correct display column. Here is the report with the newly added variable “Age” in the data:

Query: GGT value is not within the range of 0 - 57.5 when the unit is U/L and the subject was 1-15 years old when the lab was collected.

Subject	Site	visit/ Phase	Form1: Liver Tests/LIVR	Form2: Recipient Demographics / DEMO	Form3: Inclusion & Exclusion/INEX	DM Comment
10021	001	3012	GLTVAL(GGT Value U/L)=85; Age=13	DOBDT(Date of Birth)=21JUL1999	CONDATE(Consent Date)=14AUG2012	
10043	004	3028	GLTVAL(GGT Value(U/L)=59; Age=11	DOBDT(Date of Birth)=14JUL2001	CONDATE(Consent Date)=08AUG2012	

The output style can also be modified. Here is the original output; it directly listed all the variables with subset condition according to the specification:

Query: If any test is marked as clinically significant then there should be a corresponding AE; list all such tests.

Subject	Site	visit/ Phase	Form1: CHEM/ Comprehensive Chemistry	Form2: CHEM/ Comprehensive Chemistry	Form3: AEXP/ Adverse Event	DM Comment
001002	001	7000	LBDT(Date Drawn)=20FEB2011; POTCS(Potassium clinically significant)=; BUNCS(BUN clinically significant)=; CREACS(Creatinine clinically significant)=; ASTCS(AST clinically significant)=1; ALTCS(ALT clinically significant)=1; GGTCS(GGT clinically significant)=1; ALKPCS(Alk. Phos. clinically significant)=; LDHCS(LDH clinically significant)=;	LBDT(Date Drawn)=20FEB2011; TBILCS(Total Bilirubin clinically significant)=; DBILCS(Direct Bilirubin clinically significant)=; GLUCCS(Glucose clinically significant)=; SODCS(Sodium clinically significant)=0; CHLOCS(Chloride clinically significant)=; BICCS(Bicarbonate clinically significant)=; MAGNCS(Magnesium clinically significant)=; CALCCS(Calcium clinically significant)=0;	AEVTM AEVTM(Adverse Event)=Elevated AST Result; AESTDT(AE Start Date)=24JUL2006; AEENDT(AE Stop Date)=27JUL2006;	
001002	001	7000	LBDT(Date Drawn)=20FEB2011; POTCS(Potassium clinically significant)=; BUNCS(BUN clinically significant)=; CREACS(Creatinine clinically significant)=; ASTCS(AST clinically significant)=1; ALTCS(ALT clinically significant)=1; GGTCS(GGT clinically significant)=1; ALKPCS(Alk. Phos. clinically significant)=; LDHCS(LDH clinically significant)=;	LBDT(Date Drawn)=20FEB2011; TBILCS(Total Bilirubin clinically significant)=; DBILCS(Direct Bilirubin clinically significant)=; GLUCCS(Glucose clinically significant)=; SODCS(Sodium clinically significant)=0; CHLOCS(Chloride clinically significant)=; BICCS(Bicarbonate clinically significant)=; MAGNCS(Magnesium clinically significant)=; CALCCS(Calcium clinically significant)=0;	AEVTM(Adverse Event)=Vomiting; AESTDT(AE Start Date)=16NOV2006; AEENDT(AE Stop Date)=16NOV2006;	

To alter the output, after reading, subsetting, and formatting the data with the macro, an insertion of manual codes is placed before "%PrintAll". The inserted data step made three modifications here to enhance the report:

1). Added visit names to the visit numbers;
2). Printed only variables with observation as "clinically significant=1";
3). Collapsed all AEs for each subject per visit into one record.

Here is the modified output:
Query: If any test is marked as clinically significant then there should be a corresponding AE; list all such tests.

Subject	Site	visit/ Phase	Form1: CHEM/ Comprehensive Chemistry	Form2: CHEM/ Comprehensive Chemistry	Form3: AEXP/ Adverse Event	DM Comment
001002	001	7000/ Month 7	LBDT(Date Drawn)=20FEB2011; ASTCS(AST clinically significant)=1; ALTCS(ALT clinically significant)=1; GGTCS(GGT clinically significant)=1	LBDT (Date Drawn)=20FEB 2011	AEVTM(Adverse Event)=Elevated AST Result; AESTDT(AE Start Date)=24JUL2006; AEENDT(AE Stop Date)=27JUL2006; AEVTM(Adverse Event)=Vomiting; AESTDT(AE Start Date)=16NOV2006; AEENDT(AE Stop Date)=16NOV2006;	

This automation macro also offers additional flexibility for each report with an option to print all query reports, or to print the parts that are still in testing or the parts that passed validation, so that the validation process could be made more efficient.

CONCLUSION

This presentation introduces a macro that utilizes the information from the specification directly. It turns the sections in the specification into macro calls, and enables the automation process. It also provides the ultimate flexibility for the challenges encountered in our daily work. This programming strategy would be helpful in creating other macros that generate macro calls and automatic operations in any similar task that comes with a specification sheet, for example, generating some of the standard analysis datasets.

ACKNOWLEDGEMENT

Thanks for the standard template of edit checks from the Data Management team; and thanks for the Rho 20% Time project and the Rho24 project for encouraging innovation and for making these developments possible. Thanks to Rita Slater for her earlier work on codes and macros in formatting and printing the query reports. Thanks to Brandy Lind, Shayala Gibbs, and Teresa Faulkner for all their suggestions on how to improve the paper.

CONTACT INFORMATION

Christina Song
csong@rhoworld.com
SAS Programmer/analyst
Rho, Inc.
6330 Quadrangle Dr, Chapel Hill, 27514
(919) 408-8000