Paper PO-7

Not Just Another Macro

Y. Christina Song, Rho, Inc.

ABSTRACT

In most cases, macros facilitate repetitive SAS programming iterations. Even when using macros, programmers still must read specifications carefully and create macro calls accordingly. To improve overall programming efficiency substantially, here a procedure is introduced to let the specifications do the all SAS programing. Whenever possible: SAS will dynamically read specifications, tweak the data, generate all macro calls, code the text in the macros, and output data into desired reports. The methodology takes advantage of the data driven and dynamic features of SAS macros. This paper illustrates the key steps and outlines sample codes used in the programming method.

INTRODUCTION: MOST OTHER MACROS

Often SAS programmers encounter tasks directed by specifications. For such tasks, we normally have macros handle repetitive SAS programming iterations. For example, in clinical data management programming for edit-checks, macros are often used for reading in data, formatting data, and creating desired output. However, programmers still must read specifications carefully to modify data and to make macro calls accordingly. Edit-check programming in clinical data management can be time-consuming and labor intensive.

ADVANTAGES

This paper illustrates a method to let the powerful SAS all tasks. The SAS program will dynamically read specifications, tweak the data, generate all macro calls, code the text in between macros, and output data into desired reports according to the specifications. This method takes advantages of data driven and dynamic features of SAS macro. It uses common SAS procedures and requires only a few straightforward key steps. It improves overall programming efficiency significantly with following advantages:

1. It saves a substantial amount of time in programming by reading specifications with just one %include;
2. Human error can be minimized through automation.;
3. It may be possible to update specifications, program, and output instantly without even involving a programmer, to speed up the review process;
4. This programming methodology also helps to create macros which enable automation and operations planning.

KEY STEPS

Although the SAS codes for this procedure is not too complicated, the planning for overall programing strategy needs to be well thought. Here are the key steps:

1. **MAKE THE SPECIFICATIONS SAS FRIENDLY**. In most cases, the specifications are written by a non-SAS programmer. We can add another title rows using SAS code-like text. For example, original title row can be “Print records from only variable1, variable2, or variable3?” The new additional row can be “row_limit”; it will be used in the program as a variable name.

2. **TRANSLATE SPECIFICATIONS TO SAS**. Some modifications may be needed after reading the external files of specifications. For example, “scrndt should = vistdt” on the specification, will be translated with tranwrd function (varname, "should =", "NE"). So we can output any records that scrndt **NOT =** vistdt.

3. **RENAME ALL VARIABLES IN SEQUENTIAL GENERIC MANNER FOR CORRECT DATA LOGIC OPERATION**. In some cases, the specifications could call to compare two variables with the same variable name but from different data sets. The new generic variable names can handle something like this without any confusion by renaming datasets as str1, str2, str3…and variables as v1, v2, v3 etc.

4. **USE MACRO VARIABLES TO CREATE SAS MACRO CALLS**. This is the most important section in this program: SAS to create all macro calls according to the specifications. Here is an example creating one of macro calls.

First to define a variable, eg. getmydata1, in data step to store the macro call text:

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

Then to create the macro variable using above variable value and proc sql:

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

Lastly, to call the macro, simply code as:

```
&call_getmydata1;
```

So when the program is submitted, the macro-calls 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.

5. MAKE THE IN-BETWEEN-MACROS CODE AND FINALIZE THE PROGRAM

First, program all macros that are not covered in the autocall or %include. Then put the macro-calls and any in-between code in the correct sequence. So they can all work together properly.

APPLICATION

The same %include statement is needed for each query. The program could pick the correct query lists for each report from their program name. Then it will choose the correct datasets and variable names, make the correct logic operation, subset data, and output all query lists into each output report.

This program in theory should be able to handle most edit check query requests. However, some types of queries will be more straightforward than others. You can always make you own decision to cover all specifications automatically, or program the more complex queries manually.

The same idea can be used for other similar tasks. It should be able to handle all the programs directed by a table-like specification. For example, the methodology may be applicable to generating simple analysis datasets, or part of it. The purpose of this method is to let the specifications do the SAS programing whenever possible.

Contact information:
Y. Christina Song
Rho, Inc.
csong@rhoworld.com