Paper 165-30

Standardization through Macros: The Possiblilites of Using Indirect Referencing to Macro Variables

Britta Kelsey, MGI Pharma, Inc., Bloomington, MN

ABSTRACT

Have you ever wanted to write a macro and not have to specify input parameters? Have you wanted to write a global macro that can be used with different data sets without having to know the content of key variables? For example, in the pharmaceutical industry, standardization is always a goal. The same reports are repeatedly needed but the study data that is used in the reports is often slightly different with a different study design or different number of arms or cohorts. The macro facility is helpful in achieving standardization and one tool within this facility that makes this possible is indirect referencing to macro variables. It is a very useful and powerful concept but may not be well known to even the most experienced SAS® programmers. This paper uses examples to demonstrate this concept, along with how the macro processor resolves macro variables with multiple ampersands, and the use of the SYMPUT routine using DATA step variables and expressions for both arguments to produce multiple macro variables. It will show the progression of taking awkward code and condensing it with use of the SYMPUT routine and indirect referencing. All examples shown were done in the SAS system for PCs, version 8.2. The intended audience for this paper is advanced level SAS programmers.

INTRODUCTION

If you are writing a SAS macro and want to have the least amount of input parameters as possible and/or want it to work with different data sets without having to know the content of key variables, there are a few coding tricks you should be aware of. These will be addressed in this paper. They include the SYMPUT routine using DATA step variables and expressions to create multiple macro variables, indirect referencing to macro variables, and how the macro processor and symbol table work when using these techniques. I will demonstrate by example how you can take a cumbersome piece of SAS code and shorten it using the above methods. I am assuming that the reader has an understanding of Base SAS, the ODS statement, and a general knowledge of the macro facility including %LET and %DO loops.

DEFINITIONS

To understand these advanced topics, it is essential to define some macro facility elements:

Macro processor - A part of the macro facility that acts upon the symbols % and &, followed by a name token, detected during word scanning. Simply put, it recognizes a macro call and begins to execute the macro.

Symbol table - A memory location in SAS that holds the macro variable and its value.

THE SYMPUT ROUTINE

The DATA step offers functions and routines that enable transfer of information between an executing DATA step and the macro processor. One of these is the SYMPUT routine. You can use the SYMPUT routine to assign any value available to the DATA step to a macro variable. The syntax looks like:

```
call symput(macro-variable, value);
```

The *macro-variable* can be:

- A character string that is a SAS name, enclosed in quotes.
- The name of a character variable whose values are SAS names.
- A character expression that produces a macro variable name. This form is useful for creating a series of macro variables, which is the focus of this paper.

The *value* is the value you want to assign to the specific macro-variable. It can be:

- A string enclosed in quotes.
- The name of a numeric or character variable. The current value of the variable is assigned as the value of the macro variable.
- A DATA step expression. The value returned by the expression in the current observation is assigned as the value of the macro-variable.

Use the SYMPUT routine with DATA step variables or expressions for both arguments to create multiple macro variables. When the macro-variable is the name of a SAS variable or a character expression that contains a SAS variable, a unique macro variable name and value will be created from each observation in the data set. An example of this is shown in example 1 below.

EXAMPLES – CREATING MULTIPLE MACRO VARIABLES

The following examples produce listings for patients on a clinical study by the dose they are assigned to. The listings include patient number, dose level, and dosage and are saved to their own RTF file according to the dose. I want to be able to create these listings without having to look at the data to find out the dose for each dose level. It will demonstrate the need to create multiple macro variables with one SYMPUT routine.

Below is the code for the data set, called DOSES, which I will be using for examples 1-5:

```
data doses;
      input patient 1-2 dlevel $ 4-8 dose 10-12 ;
datalines;
01 dose1 50
02 dose1 50
03 dose1 50
04 dose1 50
05 dose2 100
06 dose2 100
07 dose2 100
;
run;
```

EXAMPLE 1:

Below I will create macro variables with quoting using the SYMPUT routine:

```
%let level=dose1;

data temp;
   set doses;
       if dlevel="&level";
   call symput('dose1',trim(left(dose)));
run;

ods rtf file="U:/presentations/list&dose1..rtf";
proc print data=temp;
   var patient dlevel dose;
title "Protocol 7890 - &level, &dose1 mg";
run;
ods rtf close;
```

This is a representation of the symbol table that is created from the above code:

SYMBOL TABLE	
Variable	**Value**
LEVEL	dose1
DOSE1	50

(Note: To view what the symbol table actually looks like during a SAS session, you can use the code:

```
%put _user_;
```

This will write the macro variables and their values to the log. You can also look at the data set sashelp.vmacro.)

Continue the code using dose level 2 information:

```
%let level=dose2;

data temp;
   set doses;
       if dlevel="&level";
   call symput('dose2',trim(left(dose)));
run;

ods rtf file="U:/presentations/list&dose2..rtf";
proc print data=temp;
   var patient dlevel dose;
title "Protocol 7890 - &level, &dose2 mg";
run;
ods rtf close;
```

This is a representation of the symbol table that is changed and added to from the additional set of code that includes dose level 2 information:

SYMBOL TABLE	
Variable	**Value**
LEVEL	dose2
DOSE1	50
DOSE2	100

Here is what the output file LIST50.RTF looks like:

Protocol 7890 - DOSE1,50 mg

Obs	patient	dlevel	dose
1	1	dose1	50
2	2	dose1	50
3	3	dose1	50
4	4	dose1	50

Here is what the output file LIST100.RTF looks like:

Protocol 7890 - DOSE2,100 mg

Obs	patient	dlevel	dose
1	5	dose2	100
2	6	dose2	100
3	7	dose2	100

EXAMPLE 2:

Now I will change the code slightly from example 1 to show that you can use less code by using the SYMPUT routine with variable expressions.

```
data temp;
   set doses;
   call symput(dlevel,trim(left(dose)));
```

```
run;
```

This is a representation of the symbol table that is created with just one SYMPUT routine above using the variables DLEVEL and DOSE found in the data set DOSES:

SYMBOL TABLE	
Variable	**Value**
DOSE1	50
DOSE2	100

Continuing on with the code to print the listings and save them out to an RTF file:

```
%let level=dose1;

ods rtf file="U:/presentations/list&dose1..rtf";
proc print data=temp;
       where dlevel="&level";
   var patient dlevel dose;
title "Protocol 7890 - &level, &dose1 mg";
run;
ods rtf close;

%let level=dose2;

ods rtf file="U:/presentations/list&dose2..rtf";
proc print data=temp;
       where dlevel="&level";
   var patient dlevel dose;
title "Protocol 7890 - &level, &dose2 mg";
run;
ods rtf close;
```

Looking back to example 1, example 2 has one less DATA step. However, there is still some similar code repeated twice, which is inefficient. This code could be improved by using indirect referencing to macro variables.

INDIRECT REFERENCING TO MACRO VARIABLES

You can reference macro variables indirectly using multiple ampersands for delayed resolution.

Looking back at the previous examples, you can get the value of &DOSE1 by using &LEVEL and indirect referencing. This is how:

The macro variable &LEVEL resolves to dose1.

↓

Attach & to the resolved value (&dose1)

↓

This implies you can use the reference &&LEVEL to convert the value of &LEVEL into the corresponding dosage.
This is close but there are a couple of reasons why this is not exactly correct.

There are two rules used for resolving macro variable references. The first rule is that multiple ampersands or percent signs preceding a name token cause the macro processor to rescan the reference. The second rule is that two ampersands (&&) resolve to one ampersand (&). So, use 3 ampersands (&&&) in front of a macro variable name (when its value matches the exact name of a second macro variable) to resolve to the value of the second macro variable.

EXAMPLES – INDIRECT REFERENCING BY USING MULTIPLE AMPERSANDS

Here I will expand on the examples 1 and 2 used above. I want to be able to create these listings without having to look at the data to find out how many dose levels there are in the study and the dose of each dose level. It will demonstrate the use of indirect referencing of macro variables.

EXAMPLE 3:

```
data temp;
   set doses;
   call symput(dlevel,trim(left(dose)));
run;

%let level=dose1;

ods rtf file="U:/presentations/list&&&level...rtf";
proc print data=temp;
       where dlevel="&level";
   var patient dlevel dose;
title "Protocol 7890 - &level, &&&level mg";
run;
ods rtf close;
```

Here is a visual of how the macro processor will resolve the macro variable &&&LEVEL:

$$\begin{tikzcd}\&\&\&\text{LEVEL} \arrow[r, "\text{On First Scan Resolves To}"'] & \&\text{DOSE1} \arrow[r, "\text{On Second Scan Resolves To}"'] & 50\end{tikzcd}$$

Do not use &&LEVEL because on first scan it will resolve to &LEVEL and on second scan it will resolve to dose1, which is not the desired result.

EXAMPLE 4:

This is almost the same code as in example 3, but made shorter by adding a small macro with a %DO loop. Here in the DATA step, the SUBSTR function is used with the SYMPUT routine and the variable DLEVEL to create a macro variable that resolves to the last observation, which is the last dose level. This is useful for the %DO loop that follows.

```
data temp;
   set doses end=last;
   call symput(dlevel,trim(left(dose)));
     if last then call symput('last',substr(dlevel,5));
run;
```

This is a representation of the symbol table that is created from the above code:

SYMBOL TABLE	
Variable	**Value**
DOSE1	50
DOSE2	100
LAST	2

Continuing on with the code to print the listings and save them out to an RTF file:

```
%macro doses;

%do i=1 %to &last;

%let level=dose&i;

ods rtf file="U:/presentations/list&&&level...rtf";
proc print data=temp;
       where dlevel ="&LEVEL";
   var patient dlevel dose;
```

```
title "Protocol 7890 - &level, &&&level mg";
run;
ods rtf close;

%end;

%mend doses;

%doses;
```

EXAMPLE 5:

Here is the same code as in example 4, but made shorter by getting rid of the %LET statement, replacing &LEVEL with DOSE&I, and &&&LEVEL with &&DOSE&I.

```
data temp;
   set doses end=last;
   call symput(dlevel,trim(left(dose)));
     if last then call symput('last',substr(dlevel,5));
run;

%macro doses;

%do i=1 %to &last;

ods rtf file="U:/presentations/list&&dose&i...rtf";
proc print data=temp;
       where dlevel ="dose&i";
   var patient dlevel dose;
title "Protocol 7890 - dose&i, &&dose&i mg";
run;
ods rtf close;

%end;

%mend doses;

%doses;
```

Notice that you do not need 3 ampersands in front of DOSE&I in this example. Here is a visual of how the macro processor will resolve the macro variable &&DOSE&I:

$$\begin{tikzcd} \text{\&\&DOSE\&I} \arrow[r, "\text{On First Scan Resolves To}"'] & \text{\&DOSE1} \arrow[r, "\text{On Second Scan Resolves To}"'] & 50 \end{tikzcd}$$

You can see here that using &&DOSE&I is the same as using &&&LEVEL as in example 4.

EXAMPLE – CREATING A SUMMARY TABLE WITH SPECIFIC TITLES

This example shows a macro that produces title statements with specific enrollment information about a clinical trial. These titles will be used in conjunction with a summary table created by using PROC TABULATE. The table shows best response for patients on the clinical study by the arm they are assigned to. The macro is written so you can run it without knowing how many arms there are in the study and/or what they are named. The data set for this example contains patient numbers, the study arm they are assigned to, their best response during the study, and a numeric value for their best response.

```
data arm;
      input patient 1-2 arm $ 4-5 bestres $ 6-7 bestnum 9-10;
datalines;
01 A CR 1
02 A PD 4
```

```
03 B PR 2
04 B CR 1
05 C SD 3
06 C SD 3
07 C PD 4
;
run;

data _null_;
    set arm end=last;
      if last then call symput('total',trim(left(_N_)));
run;

proc freq data=arm noprint;
   tables arm / out=a1;
run;

data a2;
   set a1 end=last;
     by arm;
   call symput('arm'||trim(left(_N_)),trim(left(arm)));
   call symput('armn'||trim(left(_N_)),trim(left(count)));
     if last then call symput('lastarm',trim(left(_N_)));
run;

%macro titlen;

 title1 "Protocol 7890";
 title2 "&total patients enrolled";
 title3
       %do i=1 %to &lastarm;
       "arm &&arm&i (N=&&armn&i.)"
      %end;
 ;

 %mend titlen;

 %titlen;
```

This is a representation of the symbol table that is created from the above code:

SYMBOL TABLE	
Variable	**Value**
TOTAL	7
ARM1	A
ARM2	B
ARM3	C
ARMN1	2
ARMN2	2
ARMN3	3
LASTARM	3

```
ods rtf file="U:/presentations/best_response.rtf";
proc sort data=arm;
       by bestnum;
run;

proc tabulate data=arm order=data;
class bestres arm;
var patient;
   table bestres=' '*patient=' ', arm='Arm'*(N colpctn='%')
   all='Total'*(N pctn='%') /rts=25 box='Best Response';
```

```
          run;
          ods rtf close;
```

Here is what the output file BEST_RESPONSE.RTF looks like:

Protocol 7890
7 patients enrolled
ARM A (N=2) ARM B (N=2) ARM C (N=3)

Best Response	Arm						Total	
	A		B		C			
	N	%	N	%	N	%	N	%
CR	1	50.00	1	50.00	.	.	2	28.57
PR	.	.	1	50.00	.	.	1	14.29
SD	.	.	.	.	2	66.67	2	28.57
PD	1	50.00	.	.	1	33.33	2	28.57

CONCLUSION

Understanding indirect referencing to macro variables and macro processing can enhance your macro writing abilities and make your macros more powerful and efficient.

REFERENCES

SAS Institute Inc. (2001), *SAS Macro Language Course Notes,* Cary, NC: SAS Institute Inc.

ACKNOWLEDGEMENTS

I would like to thank Sujata Arora, Scott Lunos, and Scott McKane for their review and helpful comments on this paper.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Please feel free to contact the author at:

Britta Kelsey
MGI Pharma, Inc.
5775 West Old Shakopee Road, Suite 100
Bloomington, MN 55437-3174
E-mail: britta.kelsey@mgipharma.com