Paper 052-29

The Power of CALL SYMPUT – DATA Step Interface by Examples

Yunchao (Susan) Tian, Social & Scientific Systems, Inc., Silver Spring, MD

ABSTRACT AND INTRODUCTION

CALL SYMPUT is a SAS® language routine that assigns a value produced in a DATA step to a macro variable. It is one of the DATA step interface tools that provides a dynamic link for communication between the SAS language and the macro facility. This paper will discuss the uses of the SYMPUT routine through real world examples. Both beginning and experienced macro programmers will benefit from the examples by recognizing the wide application and increasing the understanding of this DATA step interface tool. Besides CALL SYMPUT, other features of the macro facility that are demonstrated include the construction of macro variable names through concatenation, the double ampersand, the %DO loop, the %GOTO statement and statement label, and the implicit %EVAL function.

EXAMPLE 1: CREATE A SERIES OF VARIABLE NAMES FROM ANOTHER VARIABLE'S VALUES

When performing logistic regression, we often need to create dummy variables based on all possible values of another variable. For instance, we want to create dummy variables for the variable CON which has over 400 different integer values from 1 to 506. Basically we need to do the following:

```
IF CON = 1 THEN CON1 = 1; ELSE CON1 = 0;
IF CON = 2 THEN CON2 = 1; ELSE CON2 = 0;
. . . . . .
IF CON = 506 THEN CON506 = 1; ELSE CON506 = 0;
```

It is not practical to write this many statements. Our goal is to use the SYMPUT routine to obtain this code automatically.

In the following program, a sample data set TESTDATA with 12 observations and 1 variable is first created in step (1). Then in step (2), a data set UNIQUE is created containing 8 unique CON values. In step (3), the SYMPUT routine assigns the largest value of CON to the macro variable N. CALL SYMPUT is executed once when the DATA step reaches the end of the data set. In step (4), the macro variable N's value is retrieved and CALL SYMPUT is executed 506 times to create 506 macro variables M1-M506 with the initial value 0. The PUT function is used to eliminate a note that numeric values have been converted to character values. The LEFT function is used to left-align the value of the index variable, I, to avoid creating macro variable names with blanks. In step (5), CALL SYMPUT is executed 8 times and the values of the 8 macro variables created in step (4) are updated with the values of the corresponding CON. The 498 macro variables without the corresponding CON values will remain the initial value 0. Step (6) is a macro that generates all dummy variables for all possible values of CON. By using the %GOTO statement and statement label, the dummy variables without the corresponding CON values will not be created. Note that the double ampersand is necessary to cause the macro processor to scan the text twice first to generate the reference and then to resolve it. Step (7) invokes the macro GETCON to create the dummy variables for every observation in the data set TESTDATA. The last step prints the output data set with dummy variables shown in Table 1.

```
/* (1) Create a sample data set TESTDATA. */
DATA TESTDATA;
  INPUT CON;
  CARDS;
    1
    7
   34
  115
    7
    1
  487
   34
  506
   57
    7
   43
  ;
RUN;
```

```
/* (2) Get the unique values of CON. */
PROC SORT DATA=TESTDATA OUT=UNIQUE NODUPKEY;
  BY CON;
RUN;

/* (3) Assign the largest value of CON to the macro variable N. */
DATA _NULL_;
  SET UNIQUE END=LAST;
  IF LAST THEN CALL SYMPUT('N', PUT(CON, 3.));
RUN;

/* (4) Assign the initial value 0 to all macro variables. */
DATA _NULL_;
  DO I = 1 TO &N;
    CALL SYMPUT('M'||LEFT(PUT(I, 3.)), '0');
  END;
RUN;

/* (5) Assign the value of CON to the corresponding macro variable. */
DATA _NULL_;
  SET UNIQUE;
  CALL SYMPUT('M'||LEFT(PUT(CON, 3.)), PUT(CON, 3.));
RUN;

/* (6) Macro to generate dummy variables. */
%MACRO GETCON;
  %DO I = 1 %TO &N;
  %IF &&M&I = 0 %THEN %GOTO OUT;
    IF CON = &&M&I THEN CON&I = 1;
    ELSE CON&I = 0;
  %OUT: %END;
%MEND GETCON;

/* (7) Create dummy variables. */
DATA TESTDATA;
  SET TESTDATA;
  %GETCON
RUN;

/* (8) Print the result. */
PROC PRINT DATA=TESTDATA;
  TITLE 'Table 1. List of CON with dummy variables';
RUN;
```

Table 1. List of CON with dummy variables

Obs	CON	CON1	CON7	CON34	CON43	CON57	CON115	CON487	CON506
1	1	1	0	0	0	0	0	0	0
2	7	0	1	0	0	0	0	0	0
3	34	0	0	1	0	0	0	0	0
4	115	0	0	0	0	0	1	0	0
5	7	0	1	0	0	0	0	0	0
6	1	1	0	0	0	0	0	0	0
7	487	0	0	0	0	0	0	1	0
8	34	0	0	1	0	0	0	0	0
9	506	0	0	0	0	0	0	0	1
10	57	0	0	0	0	1	0	0	0
11	7	0	1	0	0	0	0	0	0
12	43	0	0	0	1	0	0	0	0

EXAMPLE 2: GENERATE LABELS FOR A SERIES OF VARIABLES USING EXISTING FORMATS

In this example, the problem is to label the variables FLAG1-FLAG200 using the existing formats in format library.

In the following program, a sample data set FLAGS with one observation and six variables is created. The task is to label each variable with the corresponding format, i.e., label FLAG1 with 'Red', and so on. The CALL SYMPUT statement in DATA _NULL_ step assigns the format FMTFLAG as the values to a set of macro variables FMT1-FMT6. Then in the macro LABELS, the variables FLAG1-FLAG6 are associated with the labels using the values of the macro variables FMT1-FMT6. Invoking the macro LABELS in the LABEL statement in the subsequent DATA step generates labels for all variables FLAG1-FLAG6. The PROC CONTENTS lists all variables with the assigned labels shown in Table 2.

```
DATA FLAGS;
  FLAG1 = 1; FLAG2 = 1; FLAG3 = 0;
  FLAG4 = 1; FLAG5 = 0; FLAG6 = 1;
RUN;

PROC FORMAT;
  VALUE FMTFLAG
  1='Red'
  2='Purple'
  3='Blue'
  4='Yellow'
  5='Orange'
  6='Green';
RUN;

%LET N = 6;

DATA _NULL_;
  DO I = 1 TO &N;
    CALL SYMPUT('FMT'||LEFT(PUT(I, 3.)), PUT(I, FMTFLAG.));
  END;
RUN;

%MACRO LABELS;
  %DO I = 1 %TO &N;
    FLAG&I = "&&FMT&I"
  %END;
%MEND LABELS;

DATA FLAGS;
  SET FLAGS;
  LABEL %LABELS;
RUN;

PROC CONTENTS DATA=FLAGS;
  TITLE 'Table 2. Contents of SAS data set FLAGS showing variable labels';
RUN;
```

Table 2. Contents of SAS data set FLAGS showing variable labels

#	Variable	Type	Len	Pos	Label
1	FLAG1	Num	8	0	Red
2	FLAG2	Num	8	8	Purple
3	FLAG3	Num	8	16	Blue
4	FLAG4	Num	8	24	Yellow
5	FLAG5	Num	8	32	Orange
6	FLAG6	Num	8	40	Green

EXAMPLE 3: SYMPUT ROUTINE WITH BYTE FUNCTION

We can write the following macro to read a numbered series of raw data files with the same layout to create a series of SAS data sets with the same names as the raw data files.

```
%MACRO READ(N);
  %DO I = 1 %TO &N;
    DATA CTY_&I;
      INFILE "C:\SUGI29\CTY_&I";
      INPUT ID F96 F97 F98;
    RUN;
  %END;
%MEND READ;
```

In each iteration of the %DO loop, one DATA step is generated. Invoking this macro with the parameter 3 generates three DATA steps that create three SAS data sets, CTY_1, CTY_2, and CTY_3.

Now let's suppose we want all the SAS data sets created above to be saved in an alphabetically ordered series of names. In another word, the SAS data set created from the raw data file CTY_1 will be called CTY_A, the SAS data set from CTY_2 will be called CTY_B, and so on. We can accomplish this by using the following statement:

```
CALL SYMPUT('L', BYTE(&I+64));
```

The BYTE function automatically invokes the %EVAL function and returns a character value represented by the integer value returned by the %EVAL function. For example, the BYTE function returns the character A when the value of the index variable, I, is 1 since A is the 65th character on the ASCII system. Then the SYMPUT routine assigns the character value returned by the BYTE function to the macro variable L. If we apply this statement to the DATA steps above and retrieve the macro variable L's value in subsequent DATA steps, we can create a series of SAS data sets with alphabetically ordered names. So invoking the following macro with the parameter 3 creates three SAS data sets named CTY_A, CTY_B, and CTY_C.

```
%MACRO READ(N);
  %DO I = 1 %TO &N;
    DATA CTY_&I;
      INFILE "C:\SUGI29\CTY_&I";
      INPUT ID F96 F97 F98;
      CALL SYMPUT('L', BYTE(&I+64));
    RUN;

    DATA CTY_&L;
      SET CTY_&I;
    RUN;
  %END;
%MEND READ;
```

CONCLUSION

CALL SYMPUT provides a powerful vehicle to transfer information between program steps. The examples presented in this paper can be easily modified to fit the needs of different users. There are many opportunities to apply CALL SYMPUT in our day to day programming. A few important facts need to be remembered in using CALL SYMPUT. The SYMPUT routine assigns the value of the macro variable during DATA step execution, but the macro variable references resolve during the compilation of a step or global statement. So you can't retrieve the macro variable's value in the same DATA step in which the SYMPUT routine assigns that value. The SYMPUT routine must be used as part of the CALL statement. Since CALL SYMPUT is a SAS language routine, character string arguments need to be enclosed in single or double quotation marks. In most cases, a macro variable created by the SYMPUT routine is global.

ACKNOWLEDGMENTS

I would like to thank my client Dr. Chunliu Zhan for providing me the challenges over the years, which contributed a lot to this paper. I also want to thank my colleagues Dr. Raymond Hu and Deborah Johnson for their helpful discussions.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Yunchao (Susan) Tian
Social & Scientific Systems, Inc.
8757 Georgia Avenue, 12th Floor
Silver Spring, MD 20910
Work Phone: (301) 628-3285
Fax: (301) 628-3201
Email: Stian@s-3.com