Paper BI-03-2014

Understanding Double Ampersand [&&] SAS® Macro Variables

Nina L. Werner, Madison, WI

ABSTRACT

Compound-named macro variables can be created from SAS dataset variables and resolved within a **%DO** ... **%END** loop in a macro program. This is intermediate-level content in a presentation for SAS macro beginners.

INTRODUCTION

To make a program reusable and more flexible, we can code a macro variable instead of a frequently repeated character string throughout our SAS program code. A macro variable is a character string stored in one of several macro symbol tables. It is named with 1-32 letters, digits, or underscore [_], where the first character must be underscore or a letter. A macro variable name is bracketed with ampersand [**&**] before and period [**.**] after it. While the period may be optional, it is a good practice to include it until you are able to explain why you can omit it! Avoid using SAS reserved words including '**SYS**' in macro names, since there are many SAS automatic system macro variables with names beginning with SYS available to us. Macro program code begins with **%MACRO** and the macro's name and ends with **%MEND** and the name repeated. A macro program name is invoked with percent sign [**%**] before it.

The SAS Macro facility is enabled by the **MACRO** option, which should be your SAS system default. The macro preprocessor is triggered by percent sign or ampersand [**% &**] at the beginning of a word. (This is why if we use ampersand for the logical **AND**, we need to have a space after it, although we do not need a space after carat [**^**] for the logical **NOT**.) Other options we might use to get more information about a macro program into the SAS Log are **OPTIONS MPRINT SYMBOLGEN** whose defaults are probably set off [**NOMPRINT NOSYMBOLGEN**.] Although they are beyond the scope of this paper, please turn on these options one at a time to see how they might help in understanding. Using **OPTIONS MPRINT** will write the lines of generated code in your SAS Log. **OPTIONS SYMBOLGEN** will display the value of each macro variable at the time it is resolved and executed.

CREATE AND DELETE MACRO VARIABLES

The simplest way to create and initialize a macro variable is to use **%LET** to assign a constant value to a macro name.

```
%LET macronm = abc3DefG;
%LET _tbl37 = 37 items ;
```

Notice that we do not use quotation marks around the character string. It may contain blanks. The maximum length of a **macro** variable is up to 32K and may be system dependent. The value starts with the first non-blank character after the equal sign and ends with the last non-blank character before the semicolon.

We can display text and the value of macro variables in the SAS Log with **%PUT**.

```
%PUT >>&macronm.<< {&_tbl37.} &SYSDATE. &SYSUSERID.;
```

results in this line written to the SAS Log.

```
>>abc3DefG<< {37 items} 06OCT14 NinaW
```

%PUT writes characters, so the text brackets and spaces I used "**>> << { }**" will also be displayed. This is helpful to "see" that leading and trailing blanks are excluded. We can see many other SAS system macro variables with **%PUT _AUTOMATIC_**. Macro variables may be local or global in scope. Although the distinction is beyond the scope of this paper, we can always identify the scope of a macro variable by locating it in the local or global symbol table with **%PUT _AUTOMATIC_** , **%PUT _GLOBAL_,** and **%PUT _LOCAL_**. To permanently remove macro variables, use the **%SYMDEL** statement. Notice that the macro names to be deleted are listed but are not bracketed.

```
%SYMDEL macronm _tbl37 /NOWARN;
```

CALL SYMPUTX WILL CREATE MULTIPLE MACRO VARIABLES

```
/* Example 1 CALL SYMPUTX(macrovar, value) */
/*   SAS code, this is not in a macro */

DATA _NULL_;
    SET sashelp.class END=nd;
    CALL SYMPUTX ( 'NAME' || STRIP(PUT(_N_,2.)), name );
    IF nd THEN CALL SYMPUTX ('_tot', PUT(_N_,2.));
RUN;
/* Display the macro variables we have created */
%PUT <>&_tot.<>;
%PUT _GLOBAL_;
```

In **example 1**, we use **CALL SYMPUTX** macro function to create and populate a series of 19 compound-named macro variables from two SAS dataset variables, **_N_**, the automatic data step counter, and "name" in a **DATA _NULL_** step. It has two parameters separated by a comma inside its parentheses. The first, a constant enclosed in quote marks, will be the name of the macro variable being created. When you use this name later in your SAS session, you will bracket it with [**& .**] The second is the value being assigned to that macro variable. Again, the value of any macro variable will be a character string. This value will not be available until after the execution of this **DATA _NULL_** step, but is then available for the rest of our SAS session, since this macro variable resides in the **_GLOBAL_** symbol table. Ordinary conditional logic may be used in the DATA step to define when to set the macro variable, so &_tot. is set only once by the second **CALL SYMPUTX** when the DATA step reaches the end of sashelp.class observations, _N_=19.

Let's examine the first **CALL SYMPUTX** statement in more detail. The first parameter, the macro name, is a compound construction consisting of a constant 'NAME' concatenated with a character representation of the automatic variable _N_, thus NAME1 through NAME19, since there are nineteen students in sashelp.class.

The value assigned to each macro variable is the character string from the SAS dataset variable 'name' in that observation from sashelp.class. We may need to use string functions here like PUT, STRIP, TRIM or LEFT to populate the macro variable with the character string values we ultimately will need.

This is a good time to talk about the macro preprocessor. SAS macro language writes character strings to SAS for later execution. This means that macro programs are not executed as they are encountered in our program code, but stored until SAS executes. This is important to understand, since it can lead to the most common error a new SAS macro programmer will make. We can never use the value of data in our dataset to make conditional assignments to a macro variable, since we are not reading our dataset yet at the time that the macro preprocessor is parsing our code and writing statements to be executed later. This is a powerful tool when used with more advanced programming logic.

MACRO PROGRAM WITH %DO ... %END LOOP

```
/* Example 2 %Macro Program Using %DO loop for Macro Variables */

%MACRO _all_class;
%DO i = 1 %TO &_tot.;
    TITLE "&&NAME&i.";
    PROC PRINT DATA=sashelp.class (WHERE=(name="&&NAME&i."));
        VAR age height weight;
    RUN;
%END;
%MEND _all_class;

/* Execute the macro */

%_all_class;
```

In **example 2**, we use the macro variables we created in **example 1**. We want to perform the same program steps for each of the macro variables we created, so we write a macro program containing a **%DO** ... **%END** loop which will iterate &_tot. times. Remember that &_tot. was set to the number of observations in sashelp.class. The same way we concatenated the constant 'NAME' and _N_ to create multiple macro variables, we now concatenate &name. and our loop index &i. inside a macro program. This can only work inside a macro program, not in open SAS code, because the macro preprocessor will run through the loop and write multiple iterations of SAS code to be executed later. We can run this code with **OPTIONS MPRINT** to see all the lines of SAS code generated. As we increase &i. from 1 to 19 in the loop, the double ampersand [**&&** remember? That's why we're here.] triggers the macro preprocessor to concatenate macro variable name components together, &name. with &i. thusly: &i. is first resolved to 1, then &name1. is resolved to its value, the name of student #1 = Alfred. We can run this code with **OPTIONS SYMBOLGEN** to see how the substitution is performed. We may also use **%IF** ... **%THEN** ... **%ELSE** in macro programming.

This trivial demonstration should provide a good shell framework for you to apply these elementary macro principles to your complex situations. This process is very useful, for example, in writing scheduled reports for each of the seventy-two counties in the State of Wisconsin without having to submit code seventy-two times or remember how to spell all their names.

Alfred

Obs	Age	Height	Weight
1	14	69	112.5

Alice

Obs	Age	Height	Weight
2	13	56.5	84

...

Thomas

Obs	Age	Height	Weight
18	11	57.5	85

William

Obs	Age	Height	Weight
19	15	66.5	112

Figure 1: OUTPUT (partial)

CONCLUSION

Macro variables contain character strings and are stored in symbol tables. We use macro variable names bracketed by ampersand and period. We can set their values using **%LET** or **CALL SYMPUTX**. There are many automatic SAS system macro variables available to us. A macro program can contain a **%DO** ... **%END** loop which executes multiple times. Double ampersands resolve macro variables whose names are made up of concatenated macro variable names. Remember that the SAS macro preprocessor writes SAS code that executes later, when the macro is invoked. Use **OPTIONS MPRINT** to see when and what.

KEYWORDS

```
%MACRO  %MEND  %LET  %PUT  %SYMDEL  %DO  %TO  %END
&SYSDATE.  &SYSUSERID.  CALL SYMPUTX
MPRINT  SYMBOLGEN  _AUTOMATIC_  _GLOBAL_  _LOCAL_  _NULL_
```

REFERENCES

- Carpenter, Arthur L., *Carpenter's Complete Guide to the SAS® Macro Language, 2nd Edition*, Cary, NC: SAS Institute Inc., 2004, 476 pp.
- SAS Institute Inc., *SAS® Macro Language: Reference, First Edition*, Cary, NC: SAS Institute Inc., 1997, 304 pp.

CONTACT INFORMATION

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

Name: Nina L. Werner
City, State: Madison, WI
Phone: 608 212-0689
E-mail: Ninja.Werner@me.com