Paper 142-2010

"SAS® Macros Are Just Text Substitution!" . . . ARRRRGGGHHH!!!

Dianne Piaskoski, Chinook Consulting Ltd., Richmond Hill, ON

ABSTRACT

Before I truly understood what, "SAS® macros are just text substitution!" meant, I swear that if one more person said that to me, I was going to scream. If you have heard the following, then this paper is for you:

• heard about SAS macros
• know that macros would greatly simplify your code
• understand that macros reduce the chance of errors
• do not have a resource who can explain macro basics to you before leaping too quickly into the fancier stuff.

This paper is aimed at the SAS user who has never used SAS macro variables or macro statements before, or who has been scared off by seeing too many % and & signs too quickly. By building a strong foundation in the basics of SAS macro language, you will be able to implement immediately what you have learned here. You will also gain the confidence to move forward with more advanced macro topics. I swear.

KEY WORDS

SAS macro language, macro variable, trigger, macro, input stack, global and local, macro parameters

INTRODUCTION

In the early stages of learning SAS you can easily be overwhelmed if senior SAS programmers try to show you too many shortcuts and advanced techniques before you have a firm grasp of the basics. Although having such teaching resources is invaluable, you need to be ready for them. We all learn in our own style at our own pace.

With the assumption that you are confident with the quality of your code and the accuracy of its results, I have written this paper to introduce macro variables at a very simple level so that you can use them immediately, because they are so very useful. Equally important is understanding the inner workings of the whole SAS macro facility. Although my paper does touch on this topic, for the most part it refers you to other sources. Think of the approach of this paper as learning how to drive the car before you start working on its engine.

WHAT IS A MACRO VARIABLE?

A macro variable is a tool that allows you to change your SAS code without having to search through your program for the areas that need editing. For example, suppose you wrote a program and you now run it every month. Each line of code in your program that has a reference to the current month's date must be modified. This might involve:

- changing the date in a PRINT procedure TITLE statement,

```
TITLE 'Monthly Sales Report, 30NOV2009';  1
```

- changing any of your variable or data set names such as SALES_NOV09, 2
- or changing the path of a library.

```
LIBNAME mysales 'c:\myfolder\SAS reports\SALES\NOV2009'; 3
```

You may be able to use the toolbar and do a simple <ALT> Edit/Replace/Replace All if you only have to change OCT to NOV, for example, but if there are variations of the date in your code or if you have to run each month with last year's results and the prior three months' results as well, then you are leaving yourself open to errors if you forget to make any of the required changes.

Macro variables enable you to organize and place all of these changes at the top of your program where they are easy to see and easy to edit. Equally important, macro variables make your programs flexible for other users because they do not have to search through your code to find where to make these monthly changes. Throughout the rest of your program, wherever you had hard-coded a date, you will now have place-holders that know to reference the values you have defined at the top of your program. You can create or reassign macro variables right in your DATA steps as well, and we will see this later in the paper. For now, let's start in first gear.

EXAMPLE OF USING A MACRO VARIABLE INSTEAD OF HARD-CODING

Consider the lines of code below. Even with a program as short and basic as this, we will be able to see the benefits of macro variables (the programs in this paper have been tested using PC SAS 9.1.3, UNIX SAS 9.1.3, and Enterprise Guide® 4.1).

```
DATA year_1999;
   SET sashelp.orsales;
   IF year = 1999;    *give me one year at a time;
RUN;

PROC PRINT DATA = year_1999 (OBS=5);  *only print first 5 observations;
RUN;
```

Say you need to run this code for the four years, 1999 – 2002. Again, we could just use the toolbar to replace the current year with the next year or simply type over the three cases of 1999, but using a macro variable will make it robust and flexible as/if the program grows in length and complexity.

STEP 1: DEFINING A MACRO VARIABLE

At this point we need to note the difference between SAS programming language and the SAS macro language. SAS programming language is what you already know and use; the statements in DATA and PROC steps, for example. The SAS macro language has its own statements and syntax, although the two languages have many similarities

Two keyboard characters are fundamental to the macro language: the ampersand (&), and the percent sign (%). When SAS sees these two characters, or these two triggers, it knows that we are using the macro language. To define a macro variable we use the macro language's %LET statement, choose a name for the macro variable, and assign it a value.

```
%LET the_year = 1999;
```

The macro variable name is THE_YEAR and the value of the macro variable is 1999. Both are stored in the macro symbol table. Use normal SAS naming conventions for SAS macro variables (up to 32 characters, must start with a letter/underscore, and must continue with a letter/underscore/number).

STEP 2: REFERENCING THE MACRO VARIABLE

Now, instead of using 1999 in the body of our program, we are going to use the macro variable that we just defined, THE_YEAR. We precede the macro variable with the ampersand sign and replace 1999 with &THE_YEAR. The ampersand sign is the trigger that tells SAS to "enter macro mode" again. The macro processor looks in the macro symbol table for the value associated with the variable following the ampersand, THE_YEAR, and uses that value, 1999, wherever the macro variable THE_YEAR appears in your code.

The program before using a macro variable:

```
DATA year_1999;  4
   SET sashelp.orsales;
   IF year = 1999;
RUN;

PROC PRINT DATA = year_1999 (OBS=5);
RUN;
```

The program after using a macro variable:

```
%LET the_year = 1999; 5

DATA year_&the_year;  a
   SET sashelp.orsales;
   IF year = &the_year;  b
RUN;

PROC PRINT DATA = year_&the_year (OBS=5);  c
RUN;
```

At this point we are going to take a look under the hood of the car as an important distinction is necessary. The value 1999 that was assigned to the macro variable THE_YEAR is neither character nor numeric. It is just plain text. A*ll* macro variable values are *always* just storing plain text and remain as such until after the processing and substituting take place; and even then the result can still be text depending on the context and usage. If you think about it logically, the four characters 1,9,9,9 have to be text because they are:

a) replacing part of a variable name (YEAR_1999) and a variable name is just text put together to have meaning to us (a and c)

b) substituting the place-holder with text (1999) that will become the value assigned to the numeric variable YEAR.

It may seem that the value of the macro variable value is character in a) and numeric in b), but if we keep in mind that the value is text and look at the *timing* of how SAS processes the program, we should understand the difference.

Recall that when you submit a SAS program, SAS compiles the program (checks for syntax errors) and then executes it. When macro variables are part of your code then there is more involved. In a simplified explanation of the process:

1. The SAS program is submitted and the statements are held in an input stack, waiting to be processed.
2. SAS takes one statement at a time from the top of the input stack and determines the “type” of statement and where to send it.
3. If it is a regular DATA step statement then it sends the statement to the DATA step compiler where it is checked for syntax errors.
4. If the statement has a macro reference in it (i.e., macro variable, macro call, macro statement) then the statement is passed to the macro processor.
5. Depending on the macro reference, the macro processor resolves it or executes it. The result is SAS language code (i.e., looks like the SAS code you are used to writing).
6. The macro processor passes the resulting SAS code to the top of the input stack.
7. SAS continues processing from step 2.
8. When the end of the DATA step is detected the compiler sends the code to be executed.

This is what is meant by, “macros are just text substitution”. You are:

- assigning text to a macro variable
- telling the macro processor to substitute that text in the placeholders you put throughout your SAS program
- waiting for the SAS language to process the “substituted version” of your SAS program

Now when you want to run your program for the year 2000, you just change the text in the %LET statement from 1999 to 2000 and re-run the entire program – very convenient!

```
%LET the_year = 2000;

DATA year_&the_year;  d
   SET sashelp.orsales;
   IF year = &the_year;
RUN;


PROC PRINT DATA = year_&the_year (OBS=5);
RUN;
```

USING TEXT IMMEDIATELY FOLLOWING A CALL TO A MACRO VARIABLE OR, "GOTCHA" ALERT!

In the above example, we are joining the macro variable call &THE_YEAR *after* the text YEAR_ , d. If you want a macro variable reference to *precede* text then you must separate the end of it from the rest of the text with a period. Without the period the macro processor keeps reading the reference as a macro variable with one long name. For example, suppose you are creating the following data set,

```
DATA sales_1999_shoes;
```

and you want to use the same macro variable we just defined. If you write the data statement as such,

```
DATA sales_&the_year_shoes;
```

and submit it with the rest of the program you will get the following partial message in the SAS log:

```
WARNING: Apparent symbolic reference THE_YEAR_SHOES not resolved.
25   DATA sales_&the_year_shoes;
                -
                22
                 --------------
                 202
ERROR 22-322: Syntax error, expecting one of the following: a name, a quoted string,
(, /, ;, _DATA_, _LAST_, _NULL_.
```

The & trigger tells the macro processor to resolve a macro variable. The processor attempts to resolve it but the macro variable appears to be THE_YEAR_SHOES and this is not defined so we get a message saying that THE_YEAR_SHOES could not be resolved. The processor could not distinguish between the defined macro variable THE_YEAR and the rest of the text. If we put a period after &THE_YEAR then the processor knows it is the end of the macro variable's name,

```
DATA sales_&the_year._shoes;
```

There are a number of characters other than the period that denote the end of a macro variable call such as a space, semicolon, &, %, arithmetic operators, and more. With experience you will know when these characters are enough and you do not have to include the period but if in doubt, put the period in. You cannot go wrong if you always end a macro variable call with a period, although omitting it when needed *will* be problematic.

INTRODUCTION TO MACRO PROGRAMS (OR MACROS)

With such a simple program as the one above, it does not take much time or effort to next change the value of THE_YEAR to 2001 and run it, and then to 2002 and run it. This program does provide, however, an excellent stepping stone to learning about macro programs.

You may have already noticed that an iterative loop would be useful here so that we do not even have to change 1999 to 2000, and then to 2001, etc., but you cannot wrap SAS loop constructs (in this case DO / END) around DATA and PROC steps. You can, however, create a macro and wrap macro language statements (%DO / %END) around DATA and PROC steps.

CREATING A MACRO PROGRAM

You define a macro with a %MACRO statement and the name of the macro (using normal SAS naming conventions). Your already-error-free SAS code follows, and you end the macro program with %MEND statement (MEND = Macro End). You do not have to include the name of the macro in the %MEND statement, but it is good programming convention to do so as it makes your code more readable.

```
%MACRO annual_sales;

<your SAS code and/or macro code here>
```

```
%MEND annual_sales;
```

When you submit the macro, the SAS macro facility processes and compiles the macro statements and stores the macro. The macro exists for the rest of your SAS session and does not have to be resubmitted (or recompiled) again unless you change the code within the macro. At this point, the macro has not been executed. To execute the macro you call it by submitting the (trigger) percent sign and the program name, as in 6 below. NOTE: The call to a SAS macro does not end with a semicolon because it is not a SAS statement.

```
%annual_sales  6
```

If the statements within the defined macro are nothing but SAS language code then there is no point of the macro. It is when we use macro statements and/or macro functions that we start to see the flexibility and power of macros. Continuing with program 5 we will now add a macro %DO loop. Macro statement key words start with % (and what makes the code look cryptic to newer SAS users) . For illustrative purposes, we will build the macro in three steps.

STEP 1: DEFINING THE MACRO ANNUAL_SALES

As mentioned, we define the macro with a %MACRO and %MEND statement and wrap it around our program. Submitting the defined macro compiles it, submitting the statement, %ANNUAL_SALES, executes it.

```
%MACRO annual_sales;

%LET the_year = 2000;

DATA year_&the_year;
   SET sashelp.orsales;
   IF year = &the_year;
RUN;

PROC PRINT DATA = year_&the_year (OBS=5);
RUN;

%MEND annual_sales;

%annual_sales
```

STEP 2: ADDING A MACRO %DO LOOP

Instead of repeatedly changing the text value of the macro variable THE_YEAR, we remove it and add a macro %DO loop.

```
%MACRO annual_sales;  7

%DO the_year = 1999 %TO 2002;  * the_year is now an index variable AND a macro
variable ;

DATA year_&the_year;
   SET sashelp.orsales;
   IF year = &the_year;
RUN;

PROC PRINT DATA = year_&the_year (OBS=5);
RUN;

%END;

%MEND annual_sales;

%annual_sales
```

Notice how the key words of the macro statements start with the percent sign: %DO, %TO, and %END. As well, because the %DO statement is a macro statement, the index variable THE_YEAR is a macro variable. This makes it a very powerful tool. In our program in 5 we *explicitly* defined the macro variable THE_YEAR with a %LET statement.

Here we *implicitly* defined it in the %DO statement. As the loop goes through its four iterations, it generates a DATA step and a PRINT step four times.

In 7 above, THE_YEAR was defined *inside* the macro ANNUAL_SALES so it cannot be used outside of this macro. It is therefore a local macro variable; local to the macro program in which it was defined and it does not exist after the macro ANNUAL_SALES ends. In contrast, in 5, we defined THE_YEAR with a %LET statement *outside* of any macro (it was defined in open code) so it was available for use anywhere in our program (except in a DATALINES statement). THE_YEAR is a global macro variable in 5 and a local variable in 7. I leave it to the reader to explore this topic further and learn about the %GLOBAL and %LOCAL statements on his or her own.

As it stands, this macro is not a good structure because we have hard-coded values, 1999 and 2002, into our program again. We correct this by introducing macro parameters.

STEP 3: USING MACRO PARAMETERS

There are two types of macro parameters; keyword parameters and positional parameters. We will use keyword parameters and leave the learning of positional parameters to the reader.

As mentioned, we do not want values hard-coded into our macro as this is what we wanted to avoid from the beginning. Using macro parameters eliminates the need to modify the macro code. We define parameters in the %MACRO statement with their name, followed by an equal sign. The parameter name START designates the first year of annual sales that we are interested in and the parameter name STOP designates the final year. The parameters are separated by a comma. Within the macro, the names of the macro variables must be the same as the names of the parameters they are referencing. 9

It is when we call the macro that we assign values for parameters START and STOP, 10. We have used the same values as before but whenever we call the macro, we can use whatever values we want, provided they are valid values in the actual data set. The data sets SASHELP.ORSALES only has four years of data, 1999-2002, so that is the limit of our range, but we could call the macro with the values specified in 11 without any modification to the macro.

```
%MACRO annual_sales( Start= , Stop= );  8

%DO j = &Start %TO &Stop;  9

DATA year_&j;
   SET sashelp.orsales;
   IF year = &j;
RUN;

PROC PRINT DATA = year_&j (obs=5);
RUN;

%END;

%MEND annual_sales;

%annual_sales( Start= 1999 , Stop= 2002 )  10

%annual_sales( Start= 2000 , Stop= 2000 )  11
```

STEP 4: RETURNING CONTROL TO THE TOP OF THE PROGRAM

Suppose the above macro is just one piece of code in a very long SAS program. By providing the parameter values at the end of the macro (and apart from learning a lot about macro variables and macros) we are in no better position than we were at the start of this paper because once again we have to change values that are submersed in lengthy code.

The last step in making this code truly flexible and convenient is to define macro variables at the top of our code whose values will be passed to the Start/Stop parameters. As before, we use the %LET statement. We are going to use the same names for these variables as we did for the parameters in step 3, but for illustrative purposes we will use capital letters in the macro variable names to differentiate them from the macro parameters. The final program looks like this, with lines 12 and 13 being at the very top of the program and their values being passed to line 14. We

no longer have to search through code to find where to change any values.

```
%LET START = 1999 ;   12
%LET STOP  = 2002 ;   13

------------------------ < LOTS OF OTHER CODE HERE > ------------------------

%MACRO annual_sales( Start= , Stop= );

%DO the_year = &Start %to &Stop;

DATA year_&the_year;
   SET sashelp.orsales;
   IF year = &the_year;
RUN;

PROC PRINT DATA = year_&the_year (obs=5);
RUN;

%END;

%MEND annual_sales;

%annual_sales( Start= &START , Stop= &STOP )  14
```

WORKING WITH MACRO VARIABLES AND DATE FUNCTIONS

The program we have been working with is very simple, but it still provided us with a good example for understanding the basics of macros. Because so much of what we do in the working world is based on time (daily, monthly, quarterly, etc.) we will look at how to create macro variables whose (text) values look like different dates and date formats.

If we return to the first few SAS statements at the beginning of this paper (1 - 3) we see that there are three references to a November date. We could define three macro variables and call each one as below. NOTE: If you are calling/referencing a macro variable within quotes (in a title, libname statement, or DATE9. format, for example), you *must* use double quotes. A macro variable cannot be resolved if it is inside single quotes.

```
%LET date1 = 30NOV2009;  1
%LET date2 = SALES_NOV09;  2
%LET date3 = NOV2009;  3

LIBNAME mysales "c:\myfolder\SAS reports\SALES\&date3";

PROC PRINT DATA = &date2;
TITLE "Monthly Sales Report, &date1";
RUN;
```

A more reliable method involves defining one date macro variable, then letting SAS calculate and/or format the rest of the dates we want, and assigning them to macro variables. This topic is at a slightly increased level of difficulty than what was just presented, but it is *extremely* useful and flexible. It is not necessary for the reader to understand the details, but it is hoped that upon viewing the results in the LOG window, the reader will appreciate the flexibility and power of these statements.

OUTLINE

1. Define one macro variable as the reporting date (it is still a text value, it just looks like a date).
2. Convert this value to a SAS date and assign it to a macro variable.
3. Create multiple macro variables that are different dates calculated from the one reporting date in 1.
4. Create multiple macro variables that have different date formats.

THE TOOLS WE WILL USE:

1. %SYSEVALF (simplified description): a macro function that evaluates the expression in parentheses and returns a text value.

2. %SYSFUNC: a macro function that makes DATA step functions (such as PUTN and INTNX) available to the macro language (i.e., SAS macro language can "understand" SAS programming language).

3. PUTN: a SAS function that applies a numeric format to a SAS expression at run time.

4. INTNX: a date function that increments a given date/time by a specified number of intervals. The INTNX function is the "power" behind this example.

   The syntax is: INTNX('*interval*', *start-date* , *increment* , *'alignment'*) where:

   interval is a period of time such as week/month/semiyear

   start-date is our "base" date from which all others will be calculated

   increment is the number of intervals we specify between *start-date* and the date we want calculated

   alignment (optional) is a position within the interval. Possible values (positions) are BEGINNING (B), MIDDLE (M), END (E). BEGINNING is the default.

5. %PUT: a macro language statement that writes to the SAS log, akin to the PUT statement in the SAS programming language.

THE CODE

```
/** 1. Assign a date (text) to a macro variable (looks like DATE9. format) **/

%LET BaseDate   = 12NOV2009 ;

/** 2. Convert the readable date to a SAS date **/

%LET SASDate    = %SYSEVALF( "&BaseDate"d ) ;

/** 3. Define macro variables with different date formats **/

%LET Month_Year = %SYSFUNC( PUTN( &SASDate , MONYY7.   ) ) ; * format NOV2009 ;
%LET Mon_Yr     = %SYSFUNC( PUTN( &SASDate , EURDFMY5. ) ) ; * format NOV09   ;
%LET Year_Mo    = %SYSFUNC( PUTN( &SASDate , YYMMN.    ) ) ; * format 200911  ;

/** 4. Define macro vars with different dates calculated from the BaseDate **/

/** 0 denotes no increment/decrement within the Month, other than going to the
END/MIDDLE of the month. **/

%LET End_Of_Month = %SYSFUNC( INTNX( Month , &SASDate , 0 , E  ) , DATE9. ) ;
%LET Mid_Of_Month = %SYSFUNC( INTNX( Month , &SASDate , 0 , M  ) , DATE9. ) ;

/** -1 denotes a decrement of one month from the BaseDate. Default is beginning of
month. 3 denotes an increment of three months from the BaseDate, at the END of the
third month. **/

%LET Prev_Month_Beg   = %SYSFUNC( INTNX( Month , &SASDate ,  -1     ) , DATE9. ) ;
%LET Next_Month_End   = %SYSFUNC( INTNX( Month , &SASDate ,   1 , E ) , DATE9. ) ;
%LET Tri_Month_End    = %SYSFUNC( INTNX( Month , &SASDate ,   3 , E ) , DATE9. ) ;
%LET Prev_Year_MidMth = %SYSFUNC( INTNX( Month , &SASDate , -12 , M ) , DATE9. ) ;
```

PRINT MACRO VARIABLES TO THE SAS LOG

```
%PUT BASEDATE = &BaseDate;

%PUT SASDATE = &SASDate;

%PUT MONTH_YEAR = &Month_Year;

%PUT MON_YR = &Mon_Yr;

%PUT YEAR_MO = &Year_Mo;

%PUT END_OF_MONTH = &End_of_Month;

%PUT MID_OF_MONTH = &Mid_of_Month;

%PUT PREV_MONTH_BEG = &Prev_Month_Beg;

%PUT NEXT_MONTH_END = &Next_Month_End;

%PUT TRI_MONTH_END = &Tri_Month_End;

%PUT PREV_YEAR_MIDMTH = &Prev_Year_MidMth;
```

RESULTS OF LOG WINDOW

```
<SNIP>

14   %PUT BASEDATE = &BASEDATE;

BASEDATE = 12NOV2009

15   %PUT SASDATE = &SASDATE;

SASDATE = 18213

16   %PUT MONTH_YEAR = &MONTH_YEAR;

MONTH_YEAR = NOV2009

17   %PUT MON_YR = &MON_YR;

MON_YR = NOV09

18   %PUT YEAR_MO = &YEAR_MO;

YEAR_MO = 200911

19   %PUT END_OF_MONTH = &END_OF_MONTH;

END_OF_MONTH = 30NOV2009

20   %PUT MID_OF_MONTH = &MID_OF_MONTH;

MID_OF_MONTH = 15NOV2009


21   %PUT PREV_MONTH_BEG = &PREV_MONTH_BEG;
```

```
PREV_MONTH_BEG = 01OCT2009

22   %PUT NEXT_MONTH_END = &NEXT_MONTH_END;

NEXT_MONTH_END = 31DEC2009

23   %PUT TRI_MONTH_END = &TRI_MONTH_END;

TRI_MONTH_END = 28FEB2010

24   %PUT PREV_YEAR_MIDMTH = &PREV_YEAR_MIDMTH;

PREV_YEAR_MIDMTH = 15NOV2008
```

We have just defined eleven macro variables. Because they were defined *outside* of a macro program they are global and can be used throughout the rest of your SAS code by calling/referencing them with &, followed by their macro variable name. If we want to write *all* macro variables to the SAS log that we (the user) have defined in a SAS session, and not just the eleven specified above, then we use the command:

```
%PUT _USER_ ;
```

CONCLUSION

In writing this paper it was difficult at times to know when to draw the line between:

- "too technical too soon",
- losing core concepts of the macro environment, and
- not providing enough challenge.

It is stressed that you understand the internals of DATA step processing (the program data vector (PDV), data descriptor, compilation, execution, etc.) before you race down the highway towards shortcuts that you do not know how to maneuver. Similarly, it is important to understand the internals of the macro facility because you may be writing macros that run without errors, but you are not getting the expected results because you do not understand the timing of how the macro references are being processed. My intent here was to get the newer SAS user's macro engine started, instill the confidence in the user to go back and "tinker under the hood", and to have her/him feel comfortable reading about more advanced macro topics. I hope this paper has transitioned you from the passenger's seat to the driver's seat on the road to using macros. Enjoy the ride.

REFERENCES

Burlew, Michele M. (2006), *SAS® Macro Programming Made Easy*. 2nd Ed., Cary NC: SAS Institute/SAS Publishing.

Carpenter, Art (2004), *Carpenter's Complete Guide to the SAS® Macro Language*. 2nd Ed., Cary NC: SAS Institute/SAS Publishing.

Droogendyk, Harry and Fecht, Marje. 2006. "Demystifying the SAS® Macro Facility – by Example." In *Proceedings of the Thirty-First Annual SAS Users Group International Conference*, San Francisco, CA, paper 251.

Dunn, Toby, 2006. "Handling Dates in the Macro Facility." In *Proceedings of the Nineteenth Annual NorthEast SAS Users Group Inc.*, Philadelphia, PA, paper cc21.

SAS OnlineDoc® 9.1.3 - HTTP://SUPPORT.SAS.COM/ONLINEDOC/913/DOCMAINPAGE.JSP

ACKNOWLEDGMENTS

Thank you to Toby Dunn and Harry Droogendyk whose valued feedback helped this to be a better paper

CONTACT INFORMATION

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

Dianne Piaskoski
Chinook Consulting Ltd.
Richmond Hill ON L4B 4H7
905-882-5966
dianne.piaskoski@gmail.com