

Paper CS-048

“SAS® macros are just text substitution!”ARRRRGGGHHH!!!!

Dianne Piaskoski, DJP SAS 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 about macros
- know that they would greatly simplify your code
- understand that they reduce the chance of errors
- do not have a resource that explains the very basics before leaping too quickly to the fancier stuff,

then this paper is for you. It 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 ‘%’s and ‘&’s too quickly. By building a strong foundation of the basics of the macro language, you will be able to implement immediately what you have learned here, as well as 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 a 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 PROC PRINT title statement,

```
title 'Monthly Sales Report, 30NOV2008'; 1
```

- changing any of your variable or data set names such as SALES_NOV08, 2
- or changing the path of a library.

```
libname mysales 'c:\myfolder\SAS reports\SALES\NOV2008'; 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 and UNIX SAS 9).

```
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

It is important to understand that there is the 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 do 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 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. All 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;
    set sashelp.orsales;
    if year = &the_year;
run;

proc print data = year_&the_year (obs=5);
run;
```

“GOTCHA” ALERT! USING TEXT IMMEDIATELY FOLLOWING A CALL TO A MACRO VARIABLE

In the above example, we are joining the macro variable call &THE_YEAR *after* the text YEAR_ . 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 reference 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 reference with a period, although omitting it when needed *will* be problematic.

INTRODUCTION TO 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 macros.

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

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 program 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 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 (for more on this, I encourage you to read (or attend!) SESUG 2008 paper/presentation SBC-126, “Which SASAUTOS Macros Are Available to My SAS® Session?”). 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 and 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 will add a macro %DO loop. In iterative loops we often use I or J for the index variable. Since we have been using the variable THE_YEAR thus far, we will use it as the index variable.

```
%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 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 for 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 into our program again, 1999 and 2002. 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 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, [8](#). 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 dataset. The datasets SASHELP.ORSALES only has years 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
```

AN EXERCISE FOR THE READER

Suppose the above macro is just one piece of code in a very large 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. Think of how you can return control so that you are specifying the values in 10 or 11 at the top of the program and referencing them in 10 or 11 without the hard-coding. You probably have already figured out that more macro variables are involved! Please feel welcome to contact me for any help with this exercise.

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 to understand 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 two ways to create macro variables whose (text) values look like different 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 separately as below. NOTE: If you are calling/referencing a macro variable within quotes (in a title or libname statement, for example), you must use double quotes. A macro variable cannot be resolved if it is inside single quotes (without discussing a more advanced topic outside the scope of this paper). For more on macro quoting, attend/read the presentation/paper, SESUG 2008: SBC-125, "Quick 'n Dirty – Small, Useful Utility Macros."

```
%let date1 = 30NOV2008; 1
%let date2 = SALES_NOV08; 2
%let date3 = NOV2008; 3

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

proc print data = &date2;
title "Monthly Sales Report, &date1";
run;
```

A more efficient method involves defining one date macro variable, letting SAS figure out the rest of the dates we want, and assigning them to macro variables. Note: This topic is at an increased level of difficulty than what you have just learned, but it is extremely useful and flexible. It is a good challenge and repetition is purposefully used throughout for reinforcement.

OUTLINE

1. Define one macro variable as the reporting month
2. Within a DATA _NULL_ step create three macro variables that are a variation of the current month
3. Create a macro variable that is one month prior to the current month
4. Create a macro variable that is one year prior to the current month

THE TOOLS WE WILL USE

1. PUT function: converts numeric variables to character
2. Date formats
3. CALL SYMPUTX: a SAS language routine that creates a macro variable and assigns it a value. The reader is directed to SAS OnlineDoc for more details and/or for the difference between SYMPUTX (SAS 9 only) and SYMPUT (SAS 8.2)
4. INTNX function: a date function that increments a given date/time by a specified interval

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

THE PROGRAM (IT MIGHT LOOK CRYPTIC BUT TAKE IT BIT BY BIT)

```
/** Assign the first of the month to a macro variable in what looks like DATE9.
format; */
%let base_date = 01NOV2008;

/** We do not want to create a dataset so use _NULL_ */
data _null_;

    /** Substitute text 01NOV2008 for &BASE_DATE, format as NOV2008, assign value
    to macro variable MONTH_YEAR */

    call symputx( 'month_year' , put( "&base_date"d , monyy7. ));

    /** Substitute text 01NOV2008 for &BASE_DATE, format as NOV08, assign value
    to macro variable MONTH_YR */

    call symputx( 'month_yr' , put( "&base_date"d , eurdfmy5. ));

    /** Substitute text 01NOV2008 for &BASE_DATE, format as 30NOV2008, 0 denotes no
    increment/decrement in the MONTH, END denotes to display end of month, assign
    value to macro variable END_OF_MONTH. */

    call symputx( 'end_of_month' , put( intnx( "month", "&base_date"d , -0, "end" )
, date9. ));

    /** Substitute text 01NOV2008 for &BASE_DATE, format as OCT2008, -1 denotes
    decrement in MONTH by 1, assign value to macro variable PREV_MONTH. */

    call symputx( 'prev_month' , put( intnx( "month", "&base_date"d , -1) , monyy7.
));

    /** Substitute text 01NOV2008 for &BASE_DATE, format as 200711, -1 denotes
    decrement in YEAR by 1, assign value to macro variable PREV_YEAR. */

    call symputx( 'prev_year' , put( intnx( "year", "&base_date"d , -1) , yymmn6.
));

run;

/** PRINT MACRO VARIABLE VALUES TO SAS LOG */

%put BASE_DATE = &base_date;

%put MONTH_YEAR = &month_year;

%put MONTH_YR = &month_yr;

%put END_OF_MONTH = &end_of_month;

%put PREV_MONTH = &prev_month;

%put PREV_YEAR = &prev_year;
```

```

/*** WHAT IS PRINTED TO THE SAS LOG WINDOW ***/

28
29  /** PRINT MACRO VARIABLE VALUES TO SAS LOG **/
30
31  %put BASE_DATE = &base_date;
BASE_DATE = 01NOV2008
32
33  %put MONTH_YEAR = &month_year;
MONTH_YEAR = NOV2008
34
35  %put MONTH_YR = &month_yr;
MONTH_YR = NOV08
36
37  %put END_OF_MONTH = &end_of_month;
END_OF_MONTH = 30NOV2008
38
39  %put PREV_MONTH = &prev_month;
PREV_MONTH = OCT2008
40
41  %put PREV_YEAR = &prev_year;
PREV_YEAR = 200701

```

We have just defined six macro variables. Because they were defined outside of a macro 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 that we have defined in a SAS session, and not just the six specified above, then use:

```
%put _USER_;
```

Although the following alternative method is more streamlined and elegant, SAS users who are newer to macros will probably be more comfortable with the method just displayed. For those who want to shift to a higher gear of macro learning I display the following code and refer you to the paper, "Handling Dates in the Macro Facility" in the references.

You will use the additional tools:

1. %SYSEVALF macro function (simplified description): evaluates the expression in parentheses and returns a text value.
2. %SYSFUNC macro function: Makes DATA step functions (such as PUTN and INTNX) available to the macro language (i.e., macro language can "understand" programming language).
3. PUTN function: Apply a numeric format to a SAS expression at run time.

```

%let BaseDate    = 01NOV2008 ;
%let SASDate     = %SysEvalF( "&BaseDate"d ) ;
%let MonthYear   = %SysFunc( PutN( &SASDate , MonYY7 ) ) ;
%let MonthYr     = %SysFunc( PutN( &SASDate , EurdMY5 ) ) ;
%let EndOfMonth  = %SysFunc( IntnX( Month , &SASDate , 0 , E ) , Date9 ) ;
%let BegOfMonth  = %SysFunc( IntnX( Month , &SASDate , 0 , B ) , Date9 ) ;
%let MidOfMonth  = %SysFunc( IntnX( Month , &SASDate , 0 , M ) , Date9 ) ;
%let PrevMonth   = %SysFunc( IntnX( Month , &SASDate , -1 , S ) , Date9 ) ;
%let NextMonth   = %SysFunc( IntnX( Month , &SASDate , 1 , S ) , Date9 ) ;
%let PrevYear    = %SysFunc( IntnX( Year , &SASDate , -1 , S ) , Date9 ) ;
%let NextYear    = %SysFunc( IntnX( Year , &SASDate , 1 , S ) , Date9 ) ;
%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 racing 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/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](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
DJP SAS Consulting Ltd.
Richmond Hill ON L4B 4H7
905-882-5966
dianne.piaskoski@gmail.com



SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.
Other brand and product names are trademarks of their respective companies.