PharmaSUG 2025 - Paper HT-012

SAS® Macro Programming Tips and Techniques

Kirk Paul Lafler, sasNerd

ABSTRACT

The SAS® Macro Language is a powerful tool for extending the capabilities of the SAS System. This presentation teaches essential macro coding concepts, programming techniques, and tips and tricks to help SAS users learn the basics of how the Macro language works. Using a collection of proven Macro Language coding techniques, attendees explore the application of how to write and process macro statements and parameters; replace text strings with macro (symbolic) variables; generate SAS code using macro techniques; manipulate macro variable values with macro functions; create and use global and local macro variables; construct logical expressions; interface the macro language with the SQL procedure; store and reuse macros; troubleshoot and debug macros; and develop efficient and portable macro language code.

INTRODUCTION

The **Macro Language** serves as an extension to the SAS System for the purpose of generating text in the form of SAS code, including partial and/or complete statements, DATA steps, PROC steps, variables, text strings, functions, informats, formats, expressions, comparison and logical operators, and other elements related to SAS syntax. As a language, the macro language provides users with its own set of statements, options, functions, as well as its own compiler.

One essential difference between macro code and SAS code is that macro code is compiled and executed before SAS DATA step and PROC step code, and the generated text is then processed by the SAS System. When programming with macro statements, the resulting program is called a MACRO. The Macro Language has its own rules for using the various statements and parameters. The Macro environment can be thought of as a lower level (3rd Generation) programming environment within the SAS System.

MACRO LANGUAGE BASICS

The macro language provides an additional set of tools to: 1) communicate between SAS steps, 2) construct executable and reusable code, 3) design custom languages, 4) develop user-friendly routines, and 5) conditionally execute DATA or PROC steps.

When a program is run, the SAS System first checks to see if a macro statement exists. If the program does not contain any macro statements, then processing continues as normal with the DATA or PROC step processor. If the program does contain one or more macro statements, then the macro processor must first execute them. The result of this execution is the production of character information, macro variables, or SAS statements, which are then be passed to the DATA or PROC step processor. The control flow of a macro process appears in Figure 1 below.

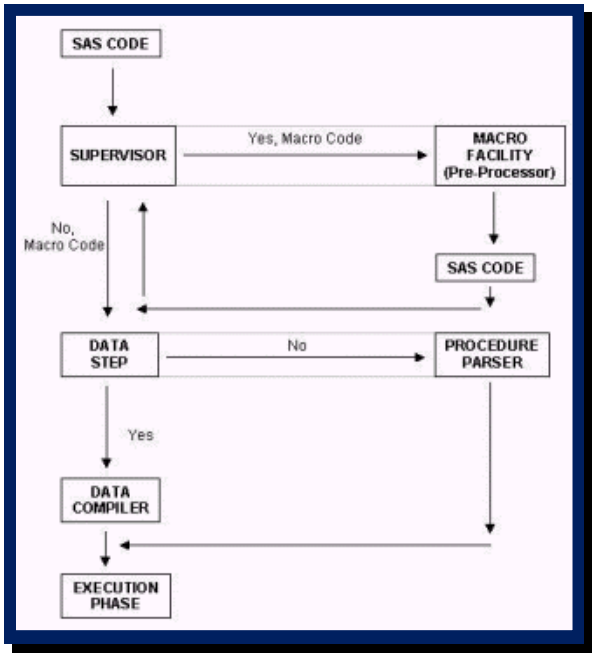

Figure 1: Macro Program Control Flow.

The SAS System Log displays information about the compilation and execution of a SAS program. This information is a vital part of any SAS execution which when viewed provides information about: 1) What statements were executed, 2) What SAS System data sets were created, 3) The number of variables and observations each data set contains, and 4) The time and memory expended by each DATA and PROC step.

The Anatomy of a Macro

Every macro begins with a %MACRO and must contain a name for the macro. To close a macro, a %MEND is used and can optionally specify the macro name for documentation reasons. Macro text can include any of the following information:

- Constant Text
- Macro Variables
- Macro Functions
- Macro Program Statements
- Macro Expressions

Constant Text

The macro language treats constant text as character strings. Examples include:

- SAS Data Set Names
- SAS Variable Names
- SAS Statements

Macro Variables

Macro variables (symbolic variables) are not DATA step variables, but belong to the SAS System macro language. Symbolic variables, once defined, can take on many different values during the execution of a macro program. Basic rules that apply to the naming of symbolic variables are:

- A name can be one to eight characters in length
- A name must begin with a character (A-Z) or underscore (_)
- Letters, numbers, and underscores can follow the first character

Basic rules that apply to the use of symbolic variables include:

- Values range from 0 to 65,534 characters in length
- The number of characters assigned to a macro variable determines its length – no length declaration is made
- Leading and trailing blanks are not stored with the value
- May be referenced (called) inside or outside of a macro by immediately prefixing an ampersand (&) before the name
- The macro processor replaces (substitutes) the symbolic variable with the value of the symbolic variable

Examples are provided to help clarify the creation and use of macro variables.

References Inside a Macro:

```
%LET NAME=USERFILE.MASTER;
%MACRO M;
   PROC MEANS DATA=&NAME;
   RUN;
%MEND M;
```

References Outside a Macro:

```
PROC PRINT DATA=&NAME;
RUN;
```

Macro Functions

Macro functions are available to process text in macros and with macro variable values. Some macro functions are associated with DATA step functions while others are used only in the macro processor. You may notice a similarity between DATA step functions and macro functions. To illustrate how macro functions can be used, a few examples are shown below.

Examples:

```
%INDEX(argument1,argument2)

%STR(argument)

%UPCASE(argument)

%BQUOTE(argument)
```

Macro Program Statements

The macro language provides a powerful language environment for users to construct and use macro programs. There are a number of Macro program statements, many of which resemble DATA step statements in use and functionality. Macro program statements are available to instruct the macro processor what to do. Each statement begins with a percent sign (%) and is terminated with a semi-colon (;). The statements are executed by the macro processor and then passed to either the DATA or PROC step for processing.

Examples:

```
%DO;

%END;

%GLOBAL macro-variable;

%MACRO name[(parameters)/STMT];
```

Macro Expressions

Macro expressions consist of macro statements, macro variable names, constant text, and/or function names combined together. Their purpose is to tie processing operations together through the use of operators and parentheses.

Examples:

```
IF &TOTAL > 999 THEN WEIGHT=WEIGHT+1;

&CHAR = %LENGTH(&SPAN)

&COUNT = %EVAL(&COUNT + 1);
```

Tip #1 – Streamlining Command-line DMS Commands with a Macro

The macro language is a wonderful tool for streamlining frequently entered SAS Display Manager System (DMS) commands to reduce the number of keystrokes. By embedding a series of DMS commands inside a simple macro, you'll not only save by not having to enter them over and over again, but you'll improve your productivity as well. The following macro code illustrates a series of DMS commands being strung together in lieu of entering them individually on a Display Manager command line. The commands display and expand the SAS Log to full size respectively, and then position the cursor at the top of the log. Once the macro is defined, it can be called by entering %POSTSUBMIT on any DMS command line to activate the commands.

Macro Code

```
%MACRO postsubmit;
   Log;
   Clear;
   Zoom;
   Pgm;
%MEND postsubmit;
```

Tip #2 – Defining a Macro Routine with Keyword Parameters

A popular design approach for passing one or more parameters into a macro routine is with keyword parameters. A distinct advantage of keyword parameter macro routines is that default values can be assigned to each macro variable – a feature that provides users with enhanced flexibility during macro execution. Another advantage with keyword parameters is that each can be specified in any order desired when calling a macro routine – much like a SAS procedure that contains one or more keyword parameters.

To illustrate the definition of a macro routine with two keyword parameters, the following macro illustrates the display of all table names (data sets) that contain the variable TITLE in the user-assigned MYDATA libref as a cross-reference listing. To retrieve the type of information needed, you could execute multiple PROC CONTENTS against selected tables. Or in a more streamlined or efficient method, you could retrieve the information directly from the read-only Dictionary table COLUMNS with the selected columns LIBNAME, MEMNAME, NAME, TYPE and LENGTH, as shown. For more information about Dictionary tables, readers may want to view the "free" SAS Press Webinar by Kirk Paul Lafler at http://support.sas.com/publishing/bbu/webinar.html#lafler2 or the published paper by Kirk Paul Lafler, Exploring Dictionary Tables and SASHELP Views.

Macro Code

```
%MACRO COLUMNS(LIB=MYDATA, COLNAME=TITLE) ;
  PROC SQL ;
    SELECT LIBNAME, MEMNAME, NAME, TYPE, LENGTH
      FROM DICTIONARY.COLUMNS
        WHERE UPCASE(LIBNAME)="&LIB" AND
              UPCASE(NAME)="&COLNAME" AND
              UPCASE(MEMTYPE)="DATA" ;
  QUIT ;
%MEND COLUMNS ;
%COLUMNS ;
```

After Macro Resolution

```
PROC SQL ;
  SELECT LIBNAME, MEMNAME, NAME, TYPE, LENGTH
    FROM DICTIONARY.COLUMNS
      WHERE LIBNAME="MYDATA"
        AND UPCASE(NAME)="TITLE"
        AND UPCASE(MEMTYPE)="DATA" ;
QUIT ;
```

Output

Library Name	Member Name	Column Name	Column Type	Column Length
MYDATA	ACTORS	Title	char	30
MYDATA	MOVIES	Title	char	30
MYDATA	PG_MOVIES	Title	char	30
MYDATA	PG_RATED_MOVIES	Title	char	30
MYDATA	RENTAL_INFO	Title	char	30

Now let's examine another macro routine that is designed with a single keyword parameter. The following macro is designed to accept one keyword parameter called &LIB. When called, it accesses the read-only Dictionary table TABLES to display each table name and the number of observations in the user-assigned MYDATA libref. This macro provides a handy way to quickly determine the number of observations in one or all tables in a libref without having to execute multiple PROC CONTENTS by using the stored information in the Dictionary table TABLES.

Macro Code

```
%MACRO NUMROWS(LIB=MYDATA) ;
  PROC SQL ;
    SELECT LIBNAME, MEMNAME, NOBS
      FROM DICTIONARY.TABLES
        WHERE UPCASE(LIBNAME)="&LIB"
          AND UPCASE(MEMTYPE)="DATA" ;
  QUIT ;
%MEND NUMROWS ;
%NUMROWS ;
```

After Macro Resolution

```
PROC SQL;
  SELECT LIBNAME, MEMNAME, NOBS
    FROM DICTIONARY.TABLES
      WHERE LIBNAME="MYDATA"
        AND UPCASE(MEMTYPE)="DATA";
QUIT;
```

Output

Library Name	Member Name	Number of Physical Observations
MYDATA	MOVIES	22
MYDATA	CUSTOMERS	3
MYDATA	MOVIES	22
MYDATA	PATIENTS	7
MYDATA	PG_MOVIES	13
MYDATA	PG_RATED_MOVIES	13

Tip #3 – Defining a Macro Routine with Positional Parameters

The macro language also provides users with the ability to define macro routines containing positional parameters. Unlike macro routines with keyword parameters, defining "default" values with one or more positional parameters is harder to define. Instead, the assignment of values for each positional parameter is supplied at the time the macro is called – a task that may not be as user-friendly or convenient for users. It is worth noting that while any number of positional parameters can be defined for a macro routine, there should be an established limit of no more than three parameters specified using a logical and natural order. Should there be a need to define more than three macro parameters, it is recommended that a macro with keyword parameters be defined instead. Another thing to keep in mind when defining positional parameters, is that the order of the list of positional parameters must be specified in the exact order defined in the macro routine.

To illustrate the definition of a two positional parameter macro, the following macro was created to display all table names (data sets) that contain the variable TITLE in the user-assigned MYDATA libref as a cross-reference listing. To retrieve the needed type of information, you could execute multiple PROC CONTENTS against selected tables. Or in a more efficient method, you could retrieve the information directly from the read-only Dictionary table COLUMNS with the selected columns LIBNAME, MEMNAME, NAME, TYPE and LENGTH, as shown. For more information about Dictionary tables, readers may want to view the "free" SAS Press Webinar by Kirk Paul Lafler at http://support.sas.com/publishing/bbu/webinar.html#lafler2 or the published paper by Kirk Paul Lafler, Exploring Dictionary Tables and SASHELP Views.

Macro Code

```
%MACRO COLUMNS(LIB, COLNAME) ;
  PROC SQL ;
    SELECT LIBNAME, MEMNAME, NAME, TYPE, LENGTH
      FROM DICTIONARY.COLUMNS
        WHERE UPCASE(LIBNAME)="&LIB" AND
              UPCASE(NAME)="&COLNAME" AND
              UPCASE(MEMTYPE)="DATA" ;
  QUIT ;
%MEND COLUMNS ;
%COLUMNS(MYDATA,TITLE) ;
```

After Macro Resolution

```
PROC SQL ;
  SELECT LIBNAME, MEMNAME, NAME, TYPE, LENGTH
    FROM DICTIONARY.COLUMNS
      WHERE LIBNAME="MYDATA"
        AND UPCASE(NAME)="TITLE"
        AND UPCASE(MEMTYPE)="DATA" ;
QUIT ;
```

Output

Library Name	Member Name	Column Name	Column Type	Column Length
MYDATA	ACTORS	Title	char	30
MYDATA	MOVIES	Title	char	30
MYDATA	PG_MOVIES	Title	char	30
MYDATA	PG_RATED_MOVIES	Title	char	30
MYDATA	RENTAL_INFO	Title	char	30

Now let's examine another useful macro that is designed with a positional parameter. The following macro is designed to accept one positional parameter called &LIB. When called, it accesses the read-only Dictionary table TABLES to display each table name and the number of observations in the user-assigned MYDATA libref. This macro provides a handy way to quickly determine the number of observations in one or all tables in a libref without having to execute multiple PROC CONTENTS by

using the stored information in the Dictionary table TABLES.

Macro Code

```
%MACRO NUMROWS(LIB) ;
  PROC SQL ;
    SELECT LIBNAME, MEMNAME, NOBS
      FROM DICTIONARY.TABLES
        WHERE UPCASE(LIBNAME)="&LIB"
          AND UPCASE(MEMTYPE)="DATA" ;
  QUIT ;
%MEND NUMROWS ;
%NUMROWS(MYDATA) ;
```

After Macro Resolution

```
PROC SQL ;
  SELECT LIBNAME, MEMNAME, NOBS
    FROM DICTIONARY.TABLES
      WHERE LIBNAME="MYDATA"
        AND UPCASE(MEMTYPE)="DATA" ;
QUIT ;
```

Output

Library Name	Member Name	Number of Physical Observations
MYDATA	MOVIES	22
MYDATA	CUSTOMERS	3
MYDATA	MOVIES	22
MYDATA	PATIENTS	7
MYDATA	PG_MOVIES	13
MYDATA	PG_RATED_MOVIES	13

Tip #4 – Assigning a Defined Macro to a Function Key

To further reduce keystrokes and enhance user productivity even further, a call to a defined macro can be saved to a Function Key. The purpose for doing this would be to allow for one-button operation of any defined macro. To illustrate the process of saving a macro call to a Function Key, the %POSTSUBMIT macro defined in the previous tip is assigned to Function Key F12 in the KEYS window. The partial KEYS window is displayed to illustrate the process.

KEYS Window

Key	Definition
F1	help
F2	reshow
F3	end;
…	…
F10	keys
F11	command focus
F12	**%POSTSUBMIT**

Tip #5 – Referencing Macro Variables Indirectly

In each of the previous examples, a macro variable began with a single ampersand, for example, &macroname. When referenced, a macro variable defined this way is resolved using a direct approach by the macro facility to an assigned value. Although this represents the most common approach to defining and referencing a macro variable, it is not the only way a macro variable can be referenced. An alternate, and more dynamic approach supported by the macro facility is its ability to handle compound expressions consisting of a macro variable beginning with, as well as containing embedded ampersands, for example, &&TYPE&n.

Using indirect macro variable references, the next example illustrates a call to a macro containing an iterative %DO loop. The macro variable RATING1 through RATING5 contains the values G, PG, PG-13, PG-17, and R. To resolve the macro references in macro RATING, the macro processor first resolves the entire reference from left to right, resolving any pair of ampersands to a single ampersand followed by processing the next part of the reference. The macro processor then returns to the beginning of the preliminary result, resolving from left to right and continuing the process over again, as before, until all ampersands have been fully processed and the resulting macro variable produced.

Macro Code

```
%LET RATING1 = G ;
%LET RATING2 = PG ;
%LET RATING3 = PG-13 ;
%LET RATING4 = PG-17 ;
%LET RATING5 = R ;
%MACRO RATING(STOP) ;
  %DO N=1 %TO &STOP ;
      %PUT &&RATING&N ;
  %END ;
%MEND RATING ;

%RATING(3) ;
```

Output

```
G
PG
PG-13
```

Tip #6 – Debugging a Macro with SAS System Options

The SAS System offers users a number of useful system options to help debug macro issues and problems. The results associated with using macro options are automatically displayed on the SAS Log. Specific options related to macro debugging appear in alphabetical order in the following table.

SAS Option	Description
MACRO	Specifies that the macro language SYMGET and SYMPUT functions be available.
MEMERR	Controls Diagnostics.
MEMRPT	Specifies that memory usage statistics be displayed on the SAS Log.
MERROR	Presents Warning Messages when there are misspellings or when an undefined macro is called.
MLOGIC	Macro execution is traced and displayed on the SAS Log for debugging purposes.
MPRINT	SAS statements generated by macro execution are traced on the SAS Log for debugging purposes.
SYMBOLGEN	Displays text from expanding macro variables to the SAS Log.

Tip #7 – Using the Autocall Facility to Call a Macro

Macro programs can be stored as SAS programs in a location in your operating environment and called on-demand using the built-in autocall facility. Macro programs stored this way are defined once, and referenced (or called) anytime needed. This provides an effective way to store and manage your macro programs in a library aggregate. To facilitate the autocall environment, you will need to specify the SAS System options presented in the following table.

SAS Option	Description
MAUTOSOURCE	Turns on the Autocall Facility so stored macro code is included in the search for macro definitions.
MRECALL	Turns on the capability to search stored macro programs when a macro is not found.
SASAUTOS=	Specifies the location of the stored macro programs.

Tip #8 – Accessing the SAS Institute-supplied Autocall Macros

Users may be unaware that SAS Institute has provided as part of your SAS software an autocall library of existing macros. These autocall macros are automatically found in your default SASAUTOS fileref. For example, the default location of the SASAUTOS fileref under Windows XP Professional on my computer is c:\program files\sas\sas 9.1\core\sasmacro. Readers are encouraged to refer to the SAS Companion manual for the operating environment you are running under for further details.

Numerous SAS-supplied autocall macros are included – many of which act and behave as macro functions. It is worth mentioning that these autocall macros provide a wealth of effective coding techniques and can be useful as a means of improving macro coding prowess in particular for those users who learn by example. The following table depicts an alphabetical sampling of the SAS Institute-supplied autocall macros for SAS 9.1.

SASAUTOS Macro Name	SASAUTOS Macro Description
%CHNGCASE	This macro is used in the change dialog box for pmenus.
%CMPRES	This macro returns the argument passed to it in an unquoted form with multiple blanks compressed to single blanks and also with leading and trailing blanks removed.
%DATATYP	The DATATYP macro determines if the input parameter is NUMERIC or CHARacter data, and returns either CHAR or NUMERIC depending on the value passed through the parameter.
%LEFT	This macro returns the argument passed to it without any leading blanks in an unquoted form.
%LOWCASE	This macro returns the argument passed to it unchanged except that all upper-case alphabetic characters are changed to their lower-case equivalents.
%SYSRC	This macro returns a numeric value corresponding to the mnemonic string passed to it and should only be used to check return code values from SCL functions.
%TRIM	This macro returns the argument passed to it without any trailing blanks in an unquoted form.
%VERIFY	This macro returns the position of the first character in the argument that is not in the target value.

To help illustrate a SASAUTOS macro, we will display the contents of the %TRIM autocall macro below. The purpose of the %TRIM autocall macro is to remove (or trim) trailing blanks from text and return the result.

%TRIM AUTOCALL Macro

```
%macro trim(value);
%*****************************************************************;
%*  MACRO: TRIM                                                  *;
%*  USAGE: 1) %trim(argument)                                    *;
%*  DESCRIPTION:                                                 *;
%*    This macro returns the argument passed to it without any   *;
%*    trailing blanks in an unquoted form. The syntax for its use*;
%*    is similar to that of native macro functions.              *;
%*    Eg. %let macvar=%trim(&argtext)                            *;
%*                                                               *;
```

```
%*  NOTES:                                                              *;
%*    None.                                                             *;
%*********************************************************************;
  %local i;
  %do i=%length(&value) %to 1 %by -1;
    %if %qsubstr(&value,&i,1)^=%str( ) %then %goto trimmed;
  %end;
  %trimmed: %if &i>0 %then %substr(&value,1,&i);
%mend;
```

Tip #9 – Compiling a Stored Macro with the Compiled Macro Facility

A macro can be compiled once and the compiled version stored so it can be used over and over again. This approach saves time and resources because the macro does not have to be compiled each time it is called. To take advantage of this time-saving approach, you will need to either verify and/or turn on the SAS System options: MSTORED and SASMSTORE. You will also need to specify the / STORE option of the %MACRO statement. It is worth mentioning that during macro compilation only macro statements are compiled, so be aware that non-macro text and macro references are not evaluated during the compilation phase – but during macro execution.

SAS Option	Description
MSTORED	**Turns on the Compiled Macro Facility so you can take advantage of this feature.**
SASMSTORE=	**Specifies the libref associated with the SAS catalog, SASMACR, of stored compiled macros.**

CONCLUSION

The macro language provides SAS users with a powerful language environment for constructing a library of powerful tools, routines, and reusable programs. It offers a comprehensive set of statements, options, functions, and has its own compiler. Once written and debugged macro programs can be stored in a location on your operating environment that can be referenced and accessed using an autocall macro environment. Macros can also be compiled providing for a more efficient process for executing macros because the macro does not have to be compiled over and over again. Finally, users are able to design and construct reusable macro tools that can be used again and again.

REFERENCES

Burlew, Michele M. (1998), SAS Macro Programming Made Easy, SAS Institute Inc., Cary, NC, USA.

Carpenter, Art (2004), Carpenter's Complete Guide to the SAS Macro Language, Second Edition. SAS Institute Inc., Cary, NC, USA.

Lafler, Kirk Paul (2024), *"SAS® Macro Programming Tips and Techniques,"* Proceedings of the 2024 Western Users of SAS Software (WUSS) Conference, sasNerd, Spring Valley, CA, USA.

Lafler, Kirk Paul (2019). *PROC SQL: Beyond the Basics Using SAS, Third Edition*, SAS Institute Inc., Cary, NC, USA.

Lafler, Kirk Paul (2015), *"Hands-on SAS® Macro Programming Essentials for New Users,"* Proceedings of the 2015 SAS Global Forum (SGF) Conference, Software Intelligence Corporation, Spring Valley, CA, USA.

Lafler, Kirk Paul (2013), *"Hands-on SAS® Macro Programming Tips and Techniques,"* Proceedings of the 2013 Western Users of SAS Software (WUSS) Conference, Software Intelligence Corporation, Spring Valley, CA, USA.

Lafler, Kirk Paul (2013), *"Hands-on SAS® Macro Programming Tips and Techniques,"* Proceedings of the 2013 MidWest SAS Users Group (MWSUG) Conference, Software Intelligence Corporation, Spring Valley, CA, USA.

Lafler, Kirk Paul (2013), *"Hands-on SAS® Macro Programming Tips and Techniques,"* Proceedings of the 2013 SAS Global Forum (SGF) Conference, Software Intelligence Corporation, Spring Valley, CA, USA.

Lafler, Kirk Paul (2012), *"SAS® Macro Programming Tips and Techniques,"* Proceedings of the 2012 PharmaSUG Conference, Software Intelligence Corporation, Spring Valley, CA, USA.

Lafler, Kirk Paul (2012), *"Building ReusableTools with the SAS® Macro Language,"* Proceedings of the 2012 SAS Global Forum (SGF) Conference, Software Intelligence Corporation, Spring Valley, CA, USA.

Lafler, Kirk Paul (2009), *"Building Reusable and Highly Effective Tools with the SAS® Macro Language,"* PharmaSUG 2009 Conference, Software Intelligence Corporation, Spring Valley, CA, USA.

Lafler, Kirk Paul (2008), *"Building Reusable SAS® Macro Tools,"* Michigan SAS Users Group 2008 Conference, Software Intelligence Corporation, Spring Valley, CA, USA.

Lafler, Kirk Paul (2007), *"SAS Macro Programming Tips and Techniques,"* Proceedings of the NorthEast SAS Users Group (NESUG) 2007 Conference, Software Intelligence Corporation, Spring Valley, CA, USA.

Lafler, Kirk Paul (2009), SAS System Macro Language Course Notes, Fifth Edition. Software Intelligence Corporation, Spring Valley, CA, USA.

Lafler, Kirk Paul (2007), SAS System Macro Language Course Notes, Fourth Edition. Software Intelligence Corporation, Spring Valley, CA, USA.

Lafler, Kirk Paul (2013), PROC SQL: Beyond the Basics Using SAS, Second Edition, SAS Institute Inc., Cary, NC, USA.

Roberts, Clark (1997), "Building and Using Macro Variable Lists," Proceedings of the Twenty-second Annual SAS Users Group International Conference, San Diego, CA, 441-443.

SAS Macro Language: Reference, SAS OnlineDoc® 9.2, SAS Institute Inc., Cary, NC, USA.

ACKNOWLEDGMENTS

The author thanks the PharmaSUG 2025 Conference Committee, particularly the PharmaSUG 2025 Hands-On Training (HOT) Section Chairs, Jay Iyengar and Neharika Sharma, for accepting my abstract and paper; the PharmaSUG 2025 Academic Chair, Ajay Gupta, the PharmaSUG 2025 Operations Chair, Gary Moore, for organizing and supporting a great "in-person" conference event; SAS Institute Inc. for providing SAS users with wonderful software; and SAS users everywhere for being the nicest people anywhere!

TRADEMARKS CITATIONS

AUTHOR INFORMATION

Kirk Paul Lafler is a data scientist, consultant, developer, programmer, and educator; and teaches SAS Programming and Data Management in the Statistics Department at San Diego State University. Kirk also provides project-based consulting, programming, and training services to client organizations in a variety of industries including healthcare, life sciences, credit card processing, technology, energy, research, and business; and teaches "virtual" and "live" SAS, SQL, Python, Database Management Systems (DBMS) technologies (e.g., Oracle, SQL-Server, Teradata, MySQL, MongoDB, PostgreSQL, AWS), Excel, R, cloud-based technologies, and other software and tools. Currently, Kirk serves as the Western Users of SAS Software (WUSS) Executive Committee (EC) Open-Source Advocate and Coordinator and is actively involved with several proprietary and open-source software user groups and conference committees. Kirk is the author of several books including the popular PROC SQL: Beyond the Basics Using SAS, Third Edition (SAS Press. 2019). He is also an Invited speaker, educator, keynote, and mentor; and is the recipient of 29 "Best" contributed paper, hands-on workshop (HOW), and poster awards.

Comments and suggestions are always welcome and can be sent to:

Kirk Paul Lafler, sasNerd
Consultant, Developer, Programmer, Data Scientist, Educator, and Author
Specializing in SAS® / Python / SQL / Database Management Systems / Excel / R / AWS / Cloud-based Technologies
E-mail: KirkLafler@cs.com
LinkedIn: https://www.linkedin.com/in/KirkPaulLafler/
Twitter: @sasNerd