MWSUG 2025 - Paper BB002

SAS® Macro Programming: The Basics and Beyond

Kirk Paul Lafler, sasNerd

Abstract

The SAS® Macro Language is a powerful feature for extending the capabilities of the SAS System. This paper highlights a collection of techniques for constructing reusable and effective macros tools. Attendees are introduced to the techniques associated with building functional macros that process statements containing SAS code; design reusable macro techniques; create macros containing keyword and positional parameters; utilize defensive programming tactics and techniques; build a library of macro utilities; interface the macro language with the SQL procedure; and develop efficient and portable macro language code.

Introduction

The **Macro Language** is an extension to the SAS System which provides the capability to develop SAS statement text. It consists of its own set of statements, options, functions, and has its own compiler. 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 assist in: 1) communicating between SAS steps, 2) constructing executable and reusable code, 3) designing custom languages, 4) developing 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.

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.

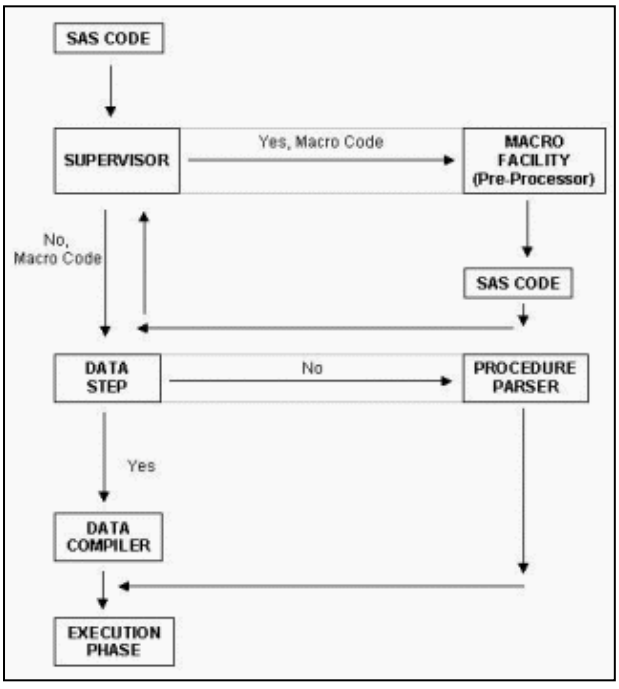

Figure 1. Macro Program Control Flow.

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

A couple examples are provided to help clarify the creation and use of macro variables.

References Inside a Macro:

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

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 – 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 #2 – 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 programs are 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 #3 – 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                                                      *;
%*                                                                   *;
%*  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)                                *;
%*********************************************************************;
%macro trim(value);
  %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 #4 – 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. This catalog stores compiled macros.

Tip #5 – 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 #6 – 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 #7 – Defining Positional Parameters

Macros are frequently designed to allow the passing of one or more parameters. This allows the creation of macro variables so text strings can be passed into the macro. The order of macro variables as positional parameters is specified when the macro is coded. The assignment of values for each positional parameter is supplied at the time the macro is called.

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 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 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 #8 – 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 #9 – Data-driven Macro and PROC SQL Programming

The SQL procedure and the macro language are two versatile tools found in the Base SAS software. Combining the two together provides users with all the tools necessary to construct highly useful and effective data-driven programs. Lafler (2018) offers a data-driven approach to creating multiple Excel files. Triggered by calling a macro to reduce coding requirements, the process uses the Macro language, PROC SQL, the ODS Excel destination, and PROC FREQ to send output (results) to Excel. The **ODS Excel** Destination became production in SAS 9.4 (M4). It serves as an interface between SAS and Excel. The ODS Excel features include:

- ✓ SAS Results and Output can be sent directly to Excel
- ✓ Offers a Flexible way to create Excel files
- ✓ Supports Reports, Tables, Statistics and Graphs
- ✓ Formats Data into Excel Worksheet cells
- ✓ Permits Automation of Production-level Workbooks.

The ODS Excel destination easily sends output and results to Excel. The ODS Excel syntax simplifies the process of sending output, reports, tables, statistics and graphs to Excel files. The ODS Excel options are able to:

- ✓ Programmatically generate output and results
- ✓ Control font used and font sizes
- ✓ Add special features to row and column headers
- ✓ Adjust row and column sizes
- ✓ Format data values
- ✓ Align data to the left, center or right
- ✓ Add hyperlinks for drill-down capability.

Producing Multiple Excel Files

In the next example, a data-driven approach using PROC SQL SELECT code embedded inside a user-defined macro routine is constructed to dynamically produce separate Excel spreadsheets containing the frequency results for each unique By-group (e.g., Movie Rating). The SELECT query processes the Movies table, creates a single-value macro variable with the number of unique movie ratings, and a value-list macro variable with a list of the unique movie ratings separated with a tilde "~". Using the FREQ procedure and a user-defined macro, both macro variables along with their respective values, an iterative macro %DO statement, a %SCAN function, and WHERE= data set option dynamically sends the results to one or more Excel spreadsheets for each By-group.

Macro and PROC SQL Code:

```
%macro multExcelfiles ;
  proc sql noprint ;
   select count(distinct rating)
    into :mrating_cnt  /* number of unique movie ratings */
     from WORK.Movies
      order by rating ;
   select distinct rating
    into :mrating_lst separated by "~"  /* list of movies */
     from WORK.Movies
      order by rating ;
  quit ;
  %do i=1 %to &mrating_cnt ;
     ods Excel file="c:/%SCAN(&mrating_lst,&i,~)_Rpt.xlsx"
              style=styles.barrettsblue
              options(embedded_titles="yes") ;
     title "%SCAN(&mrating_lst,&i,~)-Rated Movies" ;
     proc freq data=WORK.Movies(where=(rating="%SCAN(&mrating_lst,&i,~)")) ;
       tables Title ;
     run ;
     ods Excel close ;
  %end ;
  %put &mrating_lst ;
%mend multExcelfiles ;

%multExcelfiles ;
```

The dynamically generated SAS code produced by the iterative %DO statement is displayed, below.

SAS Log (Generated SAS Code):

```
ods Excel file="/folders/myfolders/G_Rpt.xlsx"
         style=styles.barrettsblue
         options(embedded_titles="yes") ;
title "G-Rated Movies" ;
proc freq data=mydata.Movies(where=(rating="G")) ;
  tables Title ;
run ;
ods Excel close ;
NOTE: There were 1 observations read from the data set MYDATA.MOVIES.
       WHERE rating='G';
NOTE: Writing EXCEL file: /folders/myfolders/G_Rpt.xlsx

ods Excel file="/folders/myfolders/PG_Rpt.xlsx"
         style=styles.barrettsblue
         options(embedded_titles="yes") ;
title "PG-Rated Movies" ;
proc freq data=mydata.Movies(where=(rating="PG")) ;
  tables Title ;
run ;
ods Excel close ;
NOTE: There were 6 observations read from the data set MYDATA.MOVIES.
       WHERE rating='PG';
NOTE: Writing EXCEL file: /folders/myfolders/PG_Rpt.xlsx

ods Excel file="/folders/myfolders/PG-13_Rpt.xlsx"
         style=styles.barrettsblue
         options(embedded_titles="yes") ;
title "PG-13-Rated Movies" ;
proc freq data=mydata.Movies(where=(rating="PG-13")) ;
  tables Title ;
run ;
ods Excel close ;
NOTE: There were 7 observations read from the data set MYDATA.MOVIES.
       WHERE rating='PG-13';
NOTE: Writing EXCEL file: /folders/myfolders/PG-13_Rpt.xlsx

ods Excel file="/folders/myfolders/R_Rpt.xlsx"
         style=styles.barrettsblue
         options(embedded_titles="yes") ;
title "R-Rated Movies" ;
proc freq data=mydata.Movies(where=(rating="R")) ;
  tables Title ;
run ;
ods Excel close ;
NOTE: There were 8 observations read from the data set MYDATA.MOVIES.
       WHERE rating='R';
NOTE: Writing EXCEL file: /folders/myfolders/R_Rpt.xlsx

G~PG~PG-13~R
```

Results (4 Excel spreadsheets are produced):

G-Rated Movies

The FREQ Procedure

Title	Frequency	Percent	Cumulative Frequency	Cumulative Percent
The Wizard of Oz	1	100.00	1	100.00

PG-Rated Movies

The FREQ Procedure

Title	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Casablanca	1	16.67	1	16.67
Jaws	1	16.67	2	33.33
Poltergeist	1	16.67	3	50.00
Rocky	1	16.67	4	66.67
Star Wars	1	16.67	5	83.33
The Hunt for Red October	1	16.67	6	100.00

PG-13-Rated Movies

The FREQ Procedure

Title	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Christmas Vacation	1	14.29	1	14.29
Forrest Gump	1	14.29	2	28.57
Ghost	1	14.29	3	42.86
Jurassic Park	1	14.29	4	57.14
Michael	1	14.29	5	71.43
National Lampoon's Vacation	1	14.29	6	85.71
Titanic	1	14.29	7	100.00

R-Rated Movies

The FREQ Procedure

Title	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Brave Heart	1	12.50	1	12.50
Coming to America	1	12.50	2	25.00
Dracula	1	12.50	3	37.50
Dressed to Kill	1	12.50	4	50.00
Lethal Weapon	1	12.50	5	62.50
Scarface	1	12.50	6	75.00
Silence of the Lambs	1	12.50	7	87.50
The Terminator	1	12.50	8	100.00

Conclusion

The SAS macro language provides users with a powerful environment for developing libraries of tools, routines, and reusable programs. It includes a rich set of statements, options, and functions, and even has its own compiler. Once written and debugged, macro programs can be stored in an autocall library, allowing them to be easily referenced and accessed across your operating environment. Macros can also be compiled for more efficient execution, since they do not need to be recompiled each time they are used. This enables users to design and build reusable macro tools that streamline programming tasks and can be leveraged repeatedly.

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 Western Users of SAS Software (WUSS) 2024 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 (2016), *"Valuable Things You Can Do with SAS DICTIONARY Tables and SASHELP Views,"* Wisconsin Illinois SAS Users (WIILSU) 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 (2006), Exploring DICTIONARY Tables and SASHELP Views, Software Intelligence Corporation, Spring Valley, CA, 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.

Acknowledgments

The author extends sincere thanks to the MWSUG 2025 Conference Committee – especially David Corliss, Academic Chair, and John LaBore, Operations Chair – for their outstanding leadership and dedication; to Louise Hadden, the Basics and Beyond Section Chair for selecting my abstract; to SAS Institute Inc. for continuing to provide SAS users with powerful and innovative software; and to SAS users everywhere for being among the kindest and most supportive people you'll ever meet!

Trademarks Citations

Author Information

Kirk Paul Lafler is a data scientist, consultant, developer, programmer, educator, and author who teaches Statistics, SAS Programming, and Data Management in the Department of Statistics at San Diego State University. Kirk also delivers project-based consulting and programming services to organizations across healthcare, life sciences, business, and other industries. As an experienced instructor, he teaches both *virtual* and *in-person* courses in SAS, SQL, Python, R, Database Management Systems (Oracle, SQL Server, Teradata, MySQL, MongoDB, PostgreSQL, AWS), Excel, cloud, and other technologies.

Kirk currently serves on the Western Users of SAS Software (WUSS) Executive Committee as the Open-Source Advocate and Coordinator, and he is actively engaged with multiple proprietary and open-source user groups and conference committees. As the author of several books, including the widely acclaimed *PROC SQL: Beyond the Basics Using SAS, Third Edition* (SAS Press, 2019), Kirk is an invited speaker, educator, keynote, mentor, and is the proud recipient of 29 "Best" contributed paper, hands-on workshop, and poster awards.

Comments and suggestions are encouraged and can be sent to:

Kirk Paul Lafler, sasNerd
Data Scientist, Consultant, Developer, Programmer, 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/