

SAS® Code and Macros: How They Interact

Bruce Gilson, Federal Reserve Board

INTRODUCTION

SAS users at every level of experience seem to struggle to understand the interaction between SAS code (DATA and PROC steps) and macros. Understanding this interaction enables SAS users to understand existing applications and to take better advantage of the tools offered by SAS software and develop code much more effectively. This paper reviews this important issue in a somewhat informal way that is similar to one-on-one discussions I have had many times with Federal Reserve Board SAS users.

Macros Generate Code

The following explanation, which is informal and over-simplified, echoes many one-on-one discussions I have had with Federal Reserve Board SAS users. It is intended to provide an intuitive understanding of macro and SAS code processing. For more information, see the examples below. For a more formal and detailed explanation, see the chapters *SAS Programs and Macro Processing* and *Macro Processing* in the *SAS 9.1.3 Macro Language Reference*.

Over 20 years ago, a long-departed SAS expert explained macro execution with a simple mantra: "Macros generate code." I still consider this an extremely helpful way to start thinking about macro execution. I think of macros as containing two things.

- Macro language elements (%DO, %IF, %PUT, %STR, etc.)
- SAS (non-macro) code

During macro execution, the macro processor executes macro language elements and generates zero or more lines of SAS code. When macro processing completes, or at any step boundary, if any lines of SAS code were generated, they are processed as if the user typed them at that location.

Understanding the interaction of SAS (DATA and PROC step) code and macros enables users to understand existing applications and easily code applications that might otherwise be much more difficult. For example, macros can be used to easily generate SAS code that can be

- a large block of code such as a DATA or PROC step or multiple steps
- a single SAS statement
- part of a SAS statement

Let's consider some examples. Examples 1 - 3 are fairly simple. Examples 4 and 5 are slightly less simple, but show an important coding concept: how to iteratively generate SAS code in a macro. Example 6 is more complex and illustrates a more automated application.

In the examples below, SAS code being "generated" means that the SAS code is treated as constant text by the macro processor, and is placed on the input stack to be processed when the macro finishes executing or at a step boundary, just as if the user had typed the code at that location in the program.

Example 1. SAS code but no macro language elements.

Macro MAC1 contains a DATA step but no macro language elements. While not a very interesting macro application in real life, this macro illustrates at the simplest level the relationship between SAS code and macros.

```
%macro mac1;
  data one;                (1)
    x1=11;                 (2)
    x2=22;                 (3)
    x3=33;                 (4)
  run;                     (5)
%mend mac1;                (6)
%mac1;
```

Macro MAC1 executes as follows.

- SAS statements (1) - (5) are generated, just as if the user had typed them at that location in the program. The following SAS statements are generated.

```
data one;
  x1=11;
  x2=22;
  x3=33;
run;
```

- (5) is a SAS statement that specifies a step boundary. It causes DATA step ONE to be compiled and executed.
- (6) is a %MEND statement that ends macro execution.

A macro of this type could be useful if the same DATA step was needed in more than one location in an application. You could just invoke the macro with %MAC1 wherever the DATA step was needed.

Example 2. SAS code and macro language elements with a step boundary.

Macro MAC2 contains a DATA step and some %LET and %PUT statements, which are macro language elements. As noted above, any SAS code generated by a macro is processed (compiled and executed) when macro processing completes or at any step boundary. This example illustrates the effect of step boundaries.

```
%macro mac2;
  data one;                                (1)
    x=1;                                    (2)
    %let blah=11;                           (3)
    %put in DATA step before PUT statement blah=&blah; (4)
    put x=;                                   (5)
    %let blah=22;                           (6)
    %put in DATA step after PUT statement blah=&blah; (7)
  run;                                       (8)

  %put after DATA step;                    (9)

%mend mac2;                                 (10)
%mac2;
```

Macro MAC2 executes as follows. Text written to the SAS log when MAC2 executes is included at the end of this section.

- SAS statements (1) and (2) are generated.
- (3) is a macro language element and is immediately executed. Macro variable BLAH is assigned the value 11.
- (4) is a macro language element and is immediately executed. The text in this statement is written to the SAS log.
- SAS statement (5) is generated.
- (6) is a macro language element and is immediately executed. Macro variable BLAH is assigned the value 11.
- (7) is a macro language element and is immediately executed. The text in this statement is written to the SAS log.
- (8) is a SAS statement that specifies a step boundary. DATA step ONE comprising statements (1), (2), (5), and (8) is compiled and executed. The PUT statement, (5), writes x=1 to the SAS log. Note in the SAS log shown below that (4) and (7) are written to the SAS log before (5).
- (9) is a macro language element and is immediately executed. The text in this statement is written to the SAS log.
- (10) is a %MEND statement that ends macro execution.

The following is written to the SAS log when MAC2 executes.

```
in DATA step before PUT statement blah=11
in DATA step after PUT statement blah=22
```

```
x=1
```

```
NOTE: The data set WORK.ONE has 1 observations and 1 variables.
```

```
NOTE: DATA statement used (Total process time):
```

```
real time      0.00 seconds
cpu time       0.01 seconds
```

```
after DATA step
```

Example 3. Use macro logic to conditionally generate a SAS statement.

In this example, we want a DATA step to include the statement X2=22; only when the value of macro variable MACVAR is 1.

DATA step when &MACVAR is 1:

```
data one;
  x1=11;
  x2=22;
  x3=33;
run;
```

DATA step when &MACVAR is not 1:

```
data one;
  x1=11;
  x3=33;
run;
```

Let's review the execution of macro MAC3 to see how it differs when &MACVAR is or is not equal to 1.

Example 3.a. When &MACVAR is equal to 1.

```
%let macvar=1;

%macro mac3;
  data one;                                (1)
    x1=11;                                  (2)
    %if &macvar=1 %then %do;                (3)
      x2=22;                                (4)
    %end;
    x3=33;                                  (5)
  run;                                       (6)
%mend mac3;                                  (7)
%mac3;
```

Macro MAC3 executes as follows.

- SAS statements (1) and (2) are generated.
- In (3), the macro processor replaces the macro variable MACVAR with its value, 1.

```
%if &macvar=1 %then %do;                (original statement)
%if 1=1 %then %do;                       (after macro variable replaced with its value)
```

1=1 is true, so the macro processor processes the code between %DO and %END. SAS statement (4) is generated.

- SAS statement (5) and (6) are generated. (6) is a step boundary, so the following SAS statements are compiled and executed.

```
data one;
  x1=11;
```

```

    x2=22;
    x3=33;
run;

```

- (7) is a %MEND statement that ends macro execution.

Example 3.b. When &MACVAR is not equal to 1.

```

%let macvar=-999;

%macro mac3;
  data one;                                (1)
  x1=11;                                    (2)
  %if &macvar=1 %then %do;                 (3)
    x2=22;                                  (4)
  %end;
  x3=33;                                    (5)
run;                                        (6)
%mend mac3;                                (7)
%mac3;

```

Macro MAC3 executes as follows.

- SAS statements (1) and (2) are generated.
- In (3), the macro processor replaces the macro variable MACVAR with its value, -999.

```

%if &macvar=1 %then %do;                    (original statement)
%if -999=1 %then %do;                       (after macro variable replaced with its value)

```

-999=1 is false, so the macro processor does not process the code between %DO and %END. SAS statement (4) is not generated.

- SAS statement (5) and (6) are generated. (6) is a step boundary, so the following SAS statements are compiled and executed.

```

data one;
  x1=11;
  x3=33;
run;

```

- (7) is a %MEND statement that ends macro execution.

Example 4. Use a macro loop to generate clauses in a SAS statement.

Data set ONE contains variables X1, X2, and X3, to be renamed to Y1, Y2, and Y3.

```

data one;
  x1=11;
  x2=22;
  x3=33;
run;

```

A DATA step to create data set TWO and rename the variables could be coded as follows.

```

data two;
  set one;
  rename
    x1=y1
    x2=y2
    x3=y3
  ;
run;

```

The same DATA step code can be generated by a macro. This is more generalized, though in a real-world application, the upper bound of the %DO loop would probably be determined dynamically in the program, not hard-coded as 3.

```

%macro mac4;
  %local j; /* local macro variable */
  data two;                                (1)
    set one;                               (2)
    rename                                 (3)
      %do j = 1 %to 3;                     (4)
        x&j = y&j                          (5)
      %end;                                (6)
  ;                                         (7)
run;                                       (8)
%mend mac4;                               (9)
%mac4;

```

The RENAME statement generated by the macro has three components.

- "RENAME" precedes the macro loop. It is generated by (3).
- Each iteration of the macro loop, (4) - (6), generates one clause of the RENAME statement.

The first time, it generates: x1 = y1
 The second time, it generates: x2 = y2
 The third time, it generates: x3 = y3

- A semicolon follows the macro loop. It is generated by (7).

Macro MAC4 executes as follows.

- SAS statements (1), (2), and (3) are generated.
- Statements (4) - (6) are executed by the macro processor as follows.
 - (4) J is replaced with its value, 1. 1 is within the loop bounds, 1 to 3, so %DO loop processing continues.
 - (5) The following SAS statement is generated: x1= y1
 - (6) %END ends the %DO statement, and the macro returns to (4).
 - (4) J is incremented from 1 to 2. 2 is within the loop bounds, 1 to 3, so %DO loop processing continues.
 - (5) The following SAS statement is generated: x2= y2
 - (6) %END ends the %DO statement, and the macro returns to (4).
 - (4) J is incremented from 2 to 3. 3 is within the loop bounds, 1 to 3, so %DO loop processing continues.
 - (5) The following SAS statement is generated: x3= y3
 - (6) %END ends the %DO statement, and the macro returns to (4).
 - (4) J is incremented from 3 to 4. 4 is outside the loop bounds, 1 to 3, so %DO loop processing ends, and processing resumes with (7).
- SAS statements (7) and (8) are generated. (7) is a semicolon that completes the RENAME statement. (8), a RUN statement, is a step boundary that ends the DATA step and causes the following SAS statements to be compiled and executed.

```

data two;
  set one;
  rename
    x1 = y1
    x2 = y2
    x3 = y3
  ;
run;

```

- (9) is a %MEND statement that ends macro execution.

This program shows that it is simple to iteratively generate part of a SAS statement in a macro once you understand the interaction between SAS code (DATA and PROC steps) and macros. This is an important technique with many possible uses.

Example 5. Use a macro loop to generate clauses in a SAS statement: an alternate approach.

An equivalent way to code Example 4 is to include just the macro loop code in the macro, as follows.

```
%macro mac5;                (1)
  %local j;                 (2)
  %do j = 1 %to 3;         (3)
    x&j = y&j              (4)
  %end;                    (5)
%mend mac5;                (6)

data two;                  (7)
  set one;                 (8)
  rename                   (9)
    %mac5                  (10)
  ;                         (11)
run;                       (12)
```

This program executes as follows.

- Statements (1) - (6) define macro MAC5. (6) ends the macro definition, and causes the macro to be compiled.
- SAS statements (7) - (9) are generated.
- Statement (10) invokes macro MAC5. The macro processor executes statements (3) - (5), inserting the resulting code as if it had been typed between (9) and (11). Execution is similar to statements (4) - (6) in Example 4, as follows.
 - (2) defines a local macro variable.
 - (3) J is replaced with its value, 1. 1 is within the loop bounds, 1 to 3, so %DO loop processing continues.
 - (4) The following SAS statement is generated: x1= y1
 - (5) %END ends the %DO statement, and the macro returns to (3).
 - (3) J is incremented from 1 to 2. 2 is within the loop bounds, 1 to 3, so %DO loop processing continues.
 - (4) The following SAS statement is generated: x2= y2
 - (5) %END ends the %DO statement, and the macro returns to (3).
 - (3) J is incremented from 2 to 3. 3 is within the loop bounds, 1 to 3, so %DO loop processing continues.
 - (4) The following SAS statement is generated: x3= y3
 - (5) %END ends the %DO statement, and the macro returns to (3).
 - (3) J is incremented from 3 to 4. 4 is outside the loop bounds, 1 to 3, so %DO loop processing ends, and processing resumes with (6).
 - (6) is a %MEND statement that ends the macro. At this point, the following SAS code has been generated, but not compiled or executed.

```
data two;
  set one;
  rename
  x1 = y1
  x2 = y2
  x3 = y3
```

- SAS statements (11) and (12) are generated. (11) is a semicolon that completes the RENAME statement. (12), a RUN statement, is a step boundary that ends the DATA step and causes the following SAS statements to be compiled and executed.

```
data two;
  set one;
  rename
```

```

x1 = y1
x2 = y2
x3 = y3
;
run;

```

The SAS statements generated and then compiled and executed in this example are the same as in Example 4.

Example 6. Use “Data driven” code to rename variables.

The final example is more complex, and illustrates a more automated application.

We want to rename all variables in a data set that start with “X” so that they are prefaced with “OLD_”. For example, rename “X” to “OLD_X” and “XYZ” to “OLD_XYZ”. We don’t know the number of variables that start with “X” or their names.

First, let’s construct a data set to work with. Three variables meet the selection criterion (start with “X”), and one does not.

```

data one;
  x1=11;
  y=22;
  xxx=33;
  xyz=44;
run;

```

Now, read the variable names that begin with "X" from a DICTIONARY table and copy them to macro variables. DICTIONARY tables are a set of read-only SAS data views that contain information about the current SAS session such as data libraries, SAS data sets, SAS macros, and external files in use or available and SAS system options settings. See the *DICTIONARY Tables* chapter in *SAS 9.1.3 Language Reference: Concepts* for an introduction to DICTIONARY tables, and *The SQL Procedure* chapter in the *Base SAS 9.1.3 Procedures Guide* for complete information. For a more detailed explanation of similar examples, see Gilson (2008).

```

proc sql noprint;                                (1)
  select count (*)
  into :num_vars
  from dictionary.columns
  where libname="WORK"
  and memname="ONE"
  and upcase(substr(name,1,1))="X";

%let num_vars=&num_vars;                          (2)

  select name                                     (3)
  into :var1-:var&num_vars
  from dictionary.columns
  where libname="WORK"
  and memname="ONE"
  and upcase(substr(name,1,1))="X";
quit;

```

This code executes as follows.

- In (1), the first SELECT statement uses the summary function COUNT to determine the number of variable names in data set WORK.ONE that begin with “X”, and copies that number to the macro variable NUM_VARS. Three variables (X1, XXX, and XYZ) meet the selection criteria, so &NUM_VARS is set to 3.
- In (2), assigning the macro variable to itself with a %LET statement removes leading and trailing blanks.
- In (3), the second SELECT statement creates three macro variables named VAR1, VAR2, and VAR3 whose values are the three variable names that met the selection criteria.

Macro variable	Value
VAR1	x1
VAR2	xxx
VAR3	xyz

Now, execute PROC DATASETS and use a RENAME statement to rename the variables. For each variable name, macro MAC6 generates the text *varname=OLD_varname*, which is used as a clause in the RENAME statement.

```

%macro mac6;                                (4)
  %local j;                                  (5)
  %do j = 1 %to &num_vars;                   (6)
    &&var&j = old_&&var&j                     (7)
  %end;                                       (8)
%mend mac6;                                  (9)

proc datasets library=work nolist;          (10)
  modify one;                                (11)
  rename                                     (12)
    %mac6                                     (13)
  ;                                           (14)
quit; run;                                   (15)

```

This program executes similarly to Example 5, as follows.

- Statements (4) - (9) define macro MAC6. (9) ends the macro definition, and causes the macro to be compiled.
- SAS statements (10) - (12) are generated.
- Statement (13) invokes macro MAC6. The macro processor executes statements as follows.
 - (5) defines a local macro variable.
 - (6) J is replaced with its value, 1. 1 is within the loop bounds, 1 to 3, so %DO loop processing continues.
 - (7) The following SAS statement is generated: `x1 = old_x1`
 - (8) %END ends the %DO statement, and the macro returns to (6).
 - (6) J is incremented from 1 to 2. 2 is within the loop bounds, 1 to 3, so %DO loop processing continues.
 - (7) The following SAS statement is generated: `xxx = old_xxx`
 - (8) %END ends the %DO statement, and the macro returns to (6).
 - (6) J is incremented from 2 to 3. 3 is within the loop bounds, 1 to 3, so %DO loop processing continues.
 - (7) The following SAS statement is generated: `xyz = old_xyz`
 - (8) %END ends the %DO statement, and the macro returns to (6).
 - (6) J is incremented from 3 to 4. 4 is outside the loop bounds, 1 to 3, so %DO loop processing ends, and macro processing resumes with (9).
 - (9) is a %MEND statement that ends the macro. At this point, the following SAS code has been generated, but not compiled or executed.

```

proc datasets library=work nolist;
  modify one;
  rename
    x1 = old_x1
    xxx = old_xxx
    xyz = old_xyz

```

- SAS statements (14) and (15) are generated. (14) is a semicolon that completes the RENAME statement, and (15) is a step boundary that ends PROC DATASETS and causes the following SAS statements to be compiled and executed.

```

proc datasets library=work nolist;
  modify one;
  rename
    x1 = old_x1
    xxx = old_xxx
    xyz = old_xyz
  ;
quit; run;

```

Note that a more robust version of this example would include the following tests.

- That no existing variable name is longer than 28 characters, since OLD_ adds 4 characters to the name and the maximum variable name length is 32 characters.
- That there are no cases where variable names OLD_ Xname and Xname both exist. That is, if the data set has variables XYZ and OLD_XYZ, the RENAME statement generates an error when it tries to rename XYZ to OLD_XYZ.

An alternate way to code this example is to copy the variables that meet the selection criterion to a single space-separated macro variable, use the automatic macro variable SQLOBS to determine the number of variable names in data set WORK.ONE that begin with "X", and use the %SCAN function to process the variable names, as follows. The PROC DATASETS step and the results are the same as above.

```
proc sql noprint;
  select name into :var_names separated by ' '
  from dictionary.columns
  where libname="WORK"
  and memname="ONE"
  and upcase (substr(name,1,1))="X";
  %let num_values=&sqlobs;
quit;

%macro mac6;
  %local j;
  %do j = 1 %to &num_values;
    %scan(&var_names, &j) = old_%scan(&var_names, &j)
  %end;
%mend mac6;

proc datasets library=work nolist;
  modify one;
  rename
    %mac6
  ;
quit; run;
```

CONCLUSION

This paper explained in a simple, informal way the interaction between SAS DATA and PROC step code and macros. Understanding this interaction allows SAS users to understand existing applications and code their own applications more effectively.

For more information, contact the author, Bruce Gilson, by mail at Federal Reserve Board, Mail Stop 157, Washington, DC 20551; by e-mail at bruce.gilson@frb.gov; or by phone at 202-452-2494.

REFERENCES

- Gilson, Bruce (2009), "Tales from the Help Desk 3: More Solutions for Simple SAS Mistakes," *Proceedings of the SAS Global Forum 2009 Conference*. <<http://support.sas.com/resources/papers/proceedings09/137-2009.pdf>>
- Gilson, Bruce (2008), "Using Data Set Values and Variable Names Outside of the DATA Step," *Proceedings of the SAS Global Forum 2008 Conference*. <<http://www2.sas.com/proceedings/forum2008/177-2008.pdf>>
- SAS Institute Inc. (2004), *Base SAS 9.1.3 Procedures Guide*, Cary, NC: SAS Institute Inc.
- SAS Institute Inc. (2004), *SAS 9.1.3 Macro Language: Reference*, Cary, NC: SAS Institute Inc.
- SAS Institute Inc. (2004), *SAS 9.1.3 Language Reference: Concepts*, Cary, NC: SAS Institute Inc.
- SAS Institute Inc. (2004), *SAS 9.1.3 Language Reference: Dictionary, Volumes 1, 2, and 3*, Cary, NC: SAS Institute Inc.

ACKNOWLEDGMENTS

The following people contributed extensively to the development of this paper: Donna Hill, Bill Jackson, and Heidi Markovitz at the Federal Reserve Board. Their support is greatly appreciated.

TRADEMARK INFORMATION

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 registered trademarks or trademarks of their respective companies.