Paper 039-31

SAS® Macro Dynamics - From Simple Basics to Powerful Invocations

Rick Andrews, Centers for Medicare and Medicaid Services, Baltimore, MD

ABSTRACT

The SAS Macro Facility offers a mechanism for expanding and customizing the functionality of the SAS System. It allows for the abbreviation of a large amount of program code conveniently and makes textual substitutions easy. The facility contains a programming language enables the execution of small sections of a program or entire steps conditionally. This paper assumes a basic knowledge of the DATA step and the use of programming logic. It will provide simple to dynamics views of the powerful capabilities of SAS macros.

INTRODUCTION

SAS macros are evaluated before compile time. When a SAS program is submitted the system routes code to the macro processor and reviews the text to see if any SAS macros or macro variables have been included. The processor cycles through the macro calls, replaces any macro references with actual values, and releases the updated program to the compiler for further processing. The resulting "effective code" will be revealed throughout the paper.

In general, the SAS macro language is portable across all operating systems with few exceptions. Typically, SAS statements that begin with a percent sign (%) are part of the macro language and macro variables can be identified by a proceeding ampersand (&).

CONTENTS

MACRO VARIABLES

Probably the simplest use of the SAS Macro Facility is the creation of macro variables. Macro variables should be used if the same value needs to be changed in more than one location each time a program is submitted. Macro variable definitions should be located at the top of a program. There are several ways to create macro variables, the easiest of which is the use of %LET.

%LET

```
DATA TEMP1;
  SET SAS.DATASET;
  WHERE CPT_CODE = '21081';
RUN;

DATA TEMP2;
  SET MY.DATA;
  WHERE CPT_CODE = '21081';
RUN;
```

Figure 1: Without Macro Variable

```
%LET CPTCODE = '21081';

DATA TEMP1;
  SET SAS.DATASET;
  WHERE CPT_CODE = &CPTCODE;
RUN;

DATA TEMP2;
  SET SAS.DATASET;
  WHERE CPT_CODE = &CPTCODE;
RUN;
```

Figure 2: With Macro Variable

Note the use of single (') quotes surrounding the value of the macro variable **CPTCODE** in the example above. When resolved, the value of **'21081'** will be passed to the compiler including the quotes. If the single quotes are omitted during the creation of the macro variable and instead are included within the **WHERE** clause, such as in the example Figure 3 below left, the SAS system will pass the literal value of **'&CPTCODE'** NOT **'21081'**. To resolve the macro variable in this manor it should be surrounded with double (") quotes as shown in Figure 4 below right.

```
%LET CPTCODE = 21081;

DATA TEMP1;
  SET SAS.DATASET;
  WHERE CPT_CODE = '&CPTCODE';
RUN;
```

Figure 3: Single Quotes

```
%LET CPTCODE = 21081;

DATA TEMP1;
  SET SAS.DATASET;
  WHERE CPT_CODE = "&CPTCODE";
RUN;
```

Figure 4: Double Quotes

PROC SQL

Another way to create a macro variable is to use the SQL procedure. This is a good way to use values from within a data set that are needed for another process. In the data set, **MY.DATA**, a given value for the **MEMBERS** variable can be placed into a macro variable depending on the requirements of a process.

In this example, the PROC SQL will grab the **FIRST** record in the data set. The macro variable **MACVAR1** will resolve to 11000.

```
PROC SQL NOPRINT;
  SELECT MEMBERS
  INTO : MACVAR1
  FROM MY.DATA;
QUIT;


%PUT &MACVAR1;
*Result = 11000;
```

Figure 5: MACVAR1

MY.DATA

YEAR	MEMBERS
1999	11000
2000	12000
2001	13000

The **%PUT** will output the results to the **LOG**.

In this example, the PROC SQL will grab the record meeting the **WHERE** criteria. The variable **MACVAR2** will resolve to 12000.

```
PROC SQL NOPRINT;
  SELECT MEMBERS
  INTO : MACVAR2
  FROM MY.DATA
  WHERE YEAR = 2000;
QUIT;

%PUT &MACVAR2;
*Result = 12000;
```

Figure 6: MACVAR2

CALL SYMPUT

The SYMPUT routine can be used to interact with the macro facility during the execution of a DATA step. The routine allows the creation of macro variables based on values within a SAS data set similar to the PROC SQL. A major difference is one process will resolve to the value of the **LAST** record in lieu of the **FIRST**!

In this example, the CALL SYMPUT will grab the **LAST** record in the table. The macro variable **MACVAR3** will resolve to 13000.

```
DATA _NULL_;
  SET MY.DATA;
  CALL SYMPUT('MACVAR3',
              MEMBERS);
RUN;


%PUT &MACVAR3;
*Result = 13000;
```

Figure 7: MACVAR3

MY.DATA

YEAR	MEMBERS
1999	11000
2000	12000
2001	13000

DATA **_NULL_** tells SAS not to create a data set.

In this example, the CALL SYMPUT will grab the record meeting the **WHERE** criteria. The variable **MACVAR4** will resolve to 12000.

```
DATA _NULL_;
  SET MY.DATA;
  WHERE YEAR = 2000;
  CALL SYMPUT('MACVAR4',
              MEMBERS);
RUN;

%PUT &MACVAR4;
*Result = 12000;
```

Figure 8: MACVAR4

DYNAMIC IN (LIST)

PROC SQL can also be used to string values together. This is particularly useful when an IN list is needed for use within a WHERE clause. Consider the data set, **MY.CPTDATA**, where a list of CPT_CODEs has been stored. This list can be placed into a macro variable and used as such: **WHERE CPT_CODE IN (&MYLIST)**.

MY.CPTDATA

CPT_CODE
21081
21082
21083
21084
21085

```
PROC SQL NOPRINT;
  SELECT "'" || CPT_CODE || "'"
  INTO : MYLIST
  SEPARATED BY ','
  FROM MY.CPTDATA;
QUIT;


%PUT &MYLIST;
*RESULT = '21081','21082','21083','21084','21085';
```

Figure 8: MYLIST

SIMPLE MACROS

Placement of SAS code within macros has various advantages. 1) Program code reduction 2) Eliminate repetitive changes 3) Provide conditional execution 4) Reduce typing errors. Below are examples of simple macro executions.

%MACRO

This simple example of a SAS macro actually does nothing more than execute the DATA step TEMP0.

```
%MACRO EXAMPLE1;

  DATA TEMP0;
     SET SAS.DATASET;
  RUN;

%MEND EXAMPLE1;

%EXAMPLE1;
```

Figure 9: Program Code EXAMPLE1

This example demonstrates how to pass values into the macro for use within the DATA step.

```
%MACRO EXAMPLE2(MACVAR,CPTCODE);

  DATA TEMP&MACVAR;
    SET SAS.DATASET;
    WHERE CPT_CODE = "&CPTCODE";
  RUN;

%MEND EXAMPLE2;

%EXAMPLE2 ( 1, 21081 );
%EXAMPLE2 ( 2, 21082 );
```

Figure 10: Program Code EXAMPLE2

In order to signify the beginning of a SAS macro the keyword **%MACRO** tells the processor the word that follows, such as **EXAMPLE1** shown in Figure 9, will be the reference used to CALL the macro later in the program. The processor keeps reading the program until it reaches the keyword, **%MEND**, at which point it knows the macro creation has ended. The syntax at right demonstrates the "effective code" created after the program has passed through the macro processor. This is the code the SAS compiler will see. Note the difference between the data set names (i.e. **TEMP1** and **TEMP2**) and the values passed into the WHERE clause.

```
DATA TEMP1;
 SET SAS.DATASET;
 WHERE CPT_CODE="21081";
RUN;
```

```
DATA TEMP2;
 SET SAS.DATASET;
 WHERE CPT_CODE="21082";
RUN;
```

Figure 11: Effective Code EXAMPLE2

%IF - %THEN - %ELSE

The macro in **EXAMPLE1** may seem somewhat superfluous for there is no need for the macro given that no values are being passed and nothing has changed. One useful purpose for creating a macro with no parameters is to conditionally execute program code. The next example shows how to check whether a file already exists before trying to create it. Checking for the existence of a file can be done for SAS data sets, libraries, directories, and external files as shown.

OPEN CODE

A statement submitted using the macro language instructs the SAS macro processor to evaluate a given operation before compile time. Some statements are only allowed inside a macro definition (between the **%MACRO** and **%MEND** statements.) There are instances when parts of the macro language can be submitted in OPEN CODE, which essentially means outside of the macro definition. Examples of this functionality include **%LET** and **%PUT**. The **%IF** and **%DO** in the example shown here cannot be executed in OPEN CODE.

```
%MACRO EXAMPLE3;

  DATA _NULL_;
    CALL SYMPUT( 'EXISTS', OPEN('TEMP1','I') );
  RUN;

  %IF NOT &EXISTS %THEN %DO;
      DATA TEMP1;
        SET SAS.DATASET;
      RUN;
  %END;
  %ELSE %DO;
      %PUT ERROR WILL ROGERS!;
  %END;

%MEND EXAMPLE3;
%EXAMPLE3;
```

OPEN - Opens SAS data set.

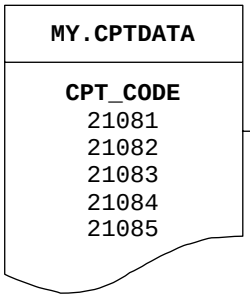
See also:

FOPEN – Opens external file.
DOPEN – Opens a directory.
FEXIST – Determine if external file exists.

Figure 12: Program Code EXAMPLE3

DYNAMIC MACROS

Creating a dynamic macro call based on values within a data set is relatively straight forward. Figure 13 below demonstrates how to use a list of codes from a data set and create a macro that will cycle for each record. The macro **EXAMPLE4** will accept two positional parameters, **MACVAR** and **CPTCODE**, same as in **EXAMPLE2** on page 2.



(1)

DATA _NULL_

The function of DATA _NULL_ in this step is to output a SAS macro call to a temporary program using the **FILE** statement. The **_N_** is the temporary variable used to count the iterations of the DATA step. Here it can be used to count observations, though not its primary function. This automatic SAS variable and the CPT_CODE variable are used within the **PUT** statement to create the appropriate number of SAS macro calls based on the information within the data set, **MY.DATA**.

%INCLUDE (2)

When the **%INCLUDE** statement is executed the syntax of the SAS program within the temporary file is read by the macro processor and the SAS macro calls are then evaluated. The subsequent "effective code" is passed to the compiler. At execution time, the system will process the 5 data steps shown below. If the number of observations within **TABLE2** changes, the macro will dynamically execute as many macro calls as becomes necessary.

```
*-----------------------------------------*;
* CREATE A MACRO TO PERFORM A GIVEN TASK   *;
*-----------------------------------------*;
%MACRO EXAMPLE4 ( MACVAR , CPTCODE );

  DATA TEMP&MACVAR; (3)
    SET SAS.DATASET;
    WHERE CPT_CODE = "&CPTCODE";
  RUN;

%MEND EXAMPLE4;

*-----------------------------------------*;
* CREATE OUTPUT FILE CONTAINING MACRO CALL *;
*-----------------------------------------*;
FILENAME TMP_FIL TEMP;

DATA _NULL_; (1)
  SET MY.DATA;
  FILE TMP_FIL;
  PUT '%EXAMPLE4(' _N_ ',' CPT_CODE ');' ;
RUN;

*-----------------------------------------*;
* INCLUDE FILE CONTAINING MACRO CALL       *;
*-----------------------------------------*;
%INCLUDE TMP_FIL; (2)
```

Figure 13: Program Code EXAMPLE4

```
%EXAMPLE4 ( 1 , 21081 );
%EXAMPLE4 ( 2 , 21082 );
%EXAMPLE4 ( 3 , 21083 );
%EXAMPLE4 ( 4 , 21084 );
%EXAMPLE4 ( 5 , 21085 );
```

(3) Figure 14: Macro Calls in Temporary File

DATA TEMP**1**; SET...; WHERE..."2108**1**"; RUN;	DATA TEMP**2**; SET...; WHERE..."2108**2**"; RUN;	DATA TEMP**3**; SET...; WHERE..."2108**3**"; RUN;	DATA TEMP**4**; SET...; WHERE..."2108**4**"; RUN;	DATA TEMP**5**; SET...; WHERE..."2108**5**"; RUN;

Figure 15: Effective Code Example4

CALL EXECUTE

The EXECUTE subroutine sends character arguments to the input stream, executes them immediately, and returns to the calling module, in this case, the DATA step. This is truly a gem of a routine as it eliminates the need for the temporary file. Note the similarity to the **PUT** statement above, the major difference being the concatenation symbols (**||**) and parenthesis that surround the statement.

```
*-----------------------------------------*;
* SUBMIT EXAMPLE5 FOR EACH CPT CODE       *;
*-----------------------------------------*;
DATA _NULL_;
  SET MY.DATA;
  CALL EXECUTE
   (
    '%EXAMPLE5(' || _N_ || ',' || CPT_CODE || ')'
   );
RUN;
```

Figure 16: Program Code CALL SYMPUT

%DO-%TO-%END

The program that follows will produce the same "effective code" as the previous two examples. The result being five DATA steps passed to the SAS compiler for processing. Here the process will dynamically **LOOP** through the data using an array of macro variables.

MY.CPTDATA

CPT_CODE
21081
21082
21083
21084
21085

```
*-----------------------------------------*;
* CREATE A MACRO TO PERFORM A GIVEN TASK   *;
*-----------------------------------------*;
%MACRO EXAMPLE6;                    (3)
                                  * NOTE THE DOUBLE AMPERSAND! *;
  DATA TEMP&I;
    SET SAS.DATASET;
    WHERE CPT_CODE = "&&MACVAR&I";
  RUN;

%MEND EXAMPLE6;

*-----------------------------------------*;
* DEFINE MACRO VARIABLES                  *;
*-----------------------------------------*;
DATA _NULL_;                        (1)
  SET MY.DATA;
  CALL SYMPUT( 'MACVAR' || LEFT(TRIM(_N_)) , CPT_CODE );
  CALL SYMPUT( 'TOTOBS' , _N_ );
RUN;

*-----------------------------------------*;
* SUBMIT EXAMPLE5 FOR EACH CPT CODE       *;
*-----------------------------------------*;
%MACRO LOOP;
  %DO I=1 %TO &TOTOBS;              (2)
    %EXAMPLE6;
  %END;
%MEND LOOP;
%LOOP;
```

Figure 17: Program Code EXAMPLE6

(1)

DATA _NULL_

The **_N_** variable is used during the creation of the **MACVAR** variables and to create the **TOTOBS** macro variable. It should be noted that _N_ should be used with caution when counting observations. If the DATA step contains a **DO** loop, _N_ may get incremented for each iteration of the loop. It can only be used as a count of records in selective cases.

During the first iteration of the DATA step, the **CALL SYMPUT** creates a macro variable called **MACVAR1** that contains the value of the CPT code for the FIRST record in the data set. Each subsequent iteration creates an additional macro variable corresponding to the record currently in memory. The "effective code" of this step is shown in Figure 18.

```
%LET MACVAR1 = 21081;
%LET MACVAR2 = 21082;
%LET MACVAR3 = 21083;
%LET MACVAR4 = 21084;
%LET MACVAR5 = 21085;
%LET TOTOBS  = 5;
```

Figure 18: Effective Code EXAMPLE6

```
%MACRO LOOP;
  %DO I=1 %TO &TOTOBS;
    %EXAMPLE6;
  %END;
%MEND LOOP;
%LOOP;
```

Figure 19: Macro LOOP

(2) **MACRO LOOP**

The SAS macro called **LOOP** has no positional parameters. Nothing is being passed into it. The **%DO-%TO-%END** is SAS macro code and is only executed from within a SAS macro. In other words, they cannot be executed in OPEN CODE. Here the process will execute the **%EXAMPLE6** macro for the total number of observations within the **TOTOBS** macro variable, 5 in this case as shown in the "effective code".

(3) **DOUBLE AMPERSAND "&&"**

The final step of the program in Figure 17 evaluates the double ampersand "**&&**". During the execution of the **%DO** LOOP, a new macro variable "**I**" was created. This macro variable contains the current iteration of the LOOP. The first time through, it evaluates to 1, then 2, and so on until it reaches 5. The evaluation of the "**&&MACVAR&I**" variable is more complicated. The result created by the double ampersand during the first iteration will be, "**&MACVAR1**", which then resolves to "**21081**". Each subsequent iteration of the **%DO** LOOP yields an increasing number, "**&MACVAR2**", "**&MACVAR3**", based on the value of "**&I**".

```
%MACRO EXAMPLE6;

  DATA TEMP&I;
    GEE  = "&&MACVAR&I";
    WHIZ = &I;
  RUN;

%MEND EXAMPLE6;
```

Figure 20: Macro EXAMPLE6

CONCLUSION

The SAS Macro Facility is a very powerful scripting language and can be employed to create very dynamic processes with ever increasing functionality. It would behoove every SAS programmer, statistician, and analyst to learn more about this robust facility. The applications shown here range from the most basic to the somewhat intricate, though only begin to scratch the surface of the capabilities.

REFERENCES

Carpenter, A. (2005), "Storing and Using a List of Values in a Macro Variable",
Proceedings of the Thirtieth Annual SAS Users Group International Conference, 30, 028-30.

Varney, B. (1999), "Creating Data Driven Programs with the Macro Language"
Proceedings of the Twenty-fourth Annual SAS Users Group International Conference, 24, 254-24.

Denis M. (2005), "CALL EXECUTE: A Powerful Data Management Tool",
Proceedings of the Thirtieth Annual SAS Users Group International Conference, 30, 027-30.

SAS Institute Inc. (1990), *SAS® Guide to Macro Processing, Version 6, Second Edition*.
Cary, NC: SAS Institute Inc.

SAS Institute Inc. 1999, *SAS®* OnlineDoc® Version Eight. "SAS Macro Language Reference."
http://v8doc.sas.com/sashtml/.

ACKNOWLEDGMENTS

The author wishes to thank the CMS SAS User Group for their support.

CONTACT INFORMATION

Your comments and questions are valued and encouraged.

Rick Andrews
Centers for Medicare and Medicaid Services
Office of Research, Development, and Information
7500 Security Boulevard
Baltimore, MD 21244
Phone: (410) 786-4088
E-mail: Richard.Andrews@cms.hhs.gov