Automated Generation of a SAS® Macro Cross-Reference Table

Connie Bryant, Computer Task Group, Inc., Indianapolis, IN

Abstract

Many sites with SAS-based processing systems utilize one or more libraries of SAS macros, either ad-hoc utility macros or the components of structured production systems. One of the challenges facing programmers is the documentation of the relationships between macros; that is, the mapping of which macros are called within each SAS program or macro (including any nested calls). This documentation can be very useful for maintaining and for learning a SAS-based system. However, it can be very tedious and labor-intensive to trace these macro calls and to represent them graphically, especially if there are frequent changes.

This poster presents an approach to the automated generation of a SAS macro cross-reference table that will document the complete set of direct and nested macro calls, in order, for each macro in a set of libraries. It provides a step-by-step explanation of the issues faced in the extraction, processing, and display of the information, with a description of the coding techniques used at each stage.

This application was written using Base SAS and the SAS Macro Language in an MVS Batch environment. Some of the processing assumes Partitioned Data Set (PDS) storage of macros but could be adapted for other platforms.

Introduction

This poster describes a set of SAS macros that were written to generate documentation of all the SAS macro calls within a set of production macro libraries. The work was performed under contract at Eli Lilly & Company. The application was inspired by the need for several analysts to learn and to document the flow of processing for a large, complex, existing system that included multiple MVS PDS libraries of macros. This application was built to be run and utilized by systems staff and was not formalized with any front-end interfaces. The code has been generalized to remove any site-specific processing and is described as though it were in one job, although in practice it would be run as at least two separate jobs.

Presentation of the Problem

In a modular, structured, environment where tested, verified code is used as much as possible, there may be a large number of macro calls within a program or macro. Documenting these calls, even when they remain simple and at one level, is tedious at best. If these macros themselves call several macros, then producing accurate, readable documentation can become very difficult. The code described in this presentation documents the macro relationships in a compact report.

There are three major obstacles to overcome when generating this automated documentation:

1) Finding the macro calls in each macro
2) Processing the macro calls found to build the expanded network of nested calls
3) Presenting the nested calls in a readable report

Figure 1 shows an example of the information to be determined and displayed for the example macros. Figure 2 shows the type of formatted listing that will be created.

ORIGINAL MACRO	MACROS CALLED WITHIN THAT MACRO
MACRO1	MACRO3
	MACRO4
	MACRO2
MACRO2	MACRO4
	MACRO3
	MACRO5
MACRO3	
MACRO4	MACRO3
	MACRO5
MACRO5	MACRO3
	MACRO3

Figure 1

ORIGINAL MACRO	CALLED MACROS: LEVEL1	LEVEL2	LEVEL3	LEVEL4
MACRO1	MACRO3			
	MACRO4	MACRO3		
	--------	MACRO5	MACRO3	
	--------	--------	MACRO3	
	MACRO2	MACRO4	MACRO3	
	--------	--------	MACRO5	MACRO3
	--------	--------	--------	MACRO3
	--------	MACRO3		
	--------	MACRO5	MACRO3	
	--------	--------	MACRO3	

Figure 2

Finding the Macro Calls - Overview

Writing the code to find the macro calls within a set of SAS code is easier than one would think. When coding macros, most programmers follow a convention of:

- having only one macro call in a line of code, and
- having the macro call be the leftmost non-blank code in a line

There are three aspects involved in parsing a module:

1) Getting the names of the macros in each library of interest
2) Parsing each macro to find potential macro calls
3) Verifying which potential calls are actual calls to macros

Initial Setup

It is necessary to process all PDS libraries that have the macros to be documented as well as the macros they may call. The filenames are set up with the convention of the name LIB + x, where x is a single letter. Macro variables are assigned to the letters and to the PDS names to facilitate dynamic generation of footnotes for the final report. The NUMLIBS variable is also assigned to allow dynamic processing later.

```
%LET LETTER1=A; %LET PDS1=USERID.SUGI22.TLIBA;
%LET LETTER2=B; %LET PDS2=USERID.SUGI22.TLIBB;
%LET NUMLIBS=2;

FILENAME LIB&LETTER1  "&PDS1" DISP=SHR;
FILENAME LIB&LETTER2  "&PDS2" DISP=SHR;

FILENAME MEMLIST '&&LIST' DISP=(NEW,DELETE)
         UNIT=SYSDA SPACE=(TRK,(5,5),RLSE)
         LRECL=80 RECFM=FB;
```

Get the List of Macros in each PDS

The %GETMACS macro generates the directory records for each PDS to be processed. The resulting list of member names from each library is used to build a SAS format and SAS data set for that library.

```
%MACRO GETMACS(INITIAL=);

  PROC SOURCE INDD=LIB&INITIAL NOPRINT
              DIRDD=MEMLIST;
  RUN;

  DATA LIB&INITIAL;
   LENGTH START $ 8  LABEL $ 3;
   RETAIN FMTNAME "$LIB&INITIAL";
   INFILE MEMLIST END=EOF;
   INPUT @1 START;
   LABEL="(&INITIAL)";
   OUTPUT;
   IF EOF THEN DO;
      START='OTHER';
      LABEL=' ';
      OUTPUT;
   END;
  RUN;

  PROC FORMAT CNTLIN=LIB&INITIAL;
  RUN;

%MEND GETMACS;

%GETMACS(INITIAL=&LETTER1)
%GETMACS(INITIAL=&LETTER2)
```

This macro creates a SAS format for each PDS, called $LIBx, that contains the member names for that library on the left side, and the initial of the library on the right side, e.g. MACRO1 = (A). When a potential macro call is found, it will be checked against all of these formats; if it is a macro, the initial of the library containing the macro will be the result of the lookup. Otherwise, the lookup will return a blank.

Find the Macros Called Within Each Macro

The principle used in parsing for potential macros is straightforward: If a line of code contains

%XXXX(or **%XXXX**

it may be a macro call. This potential macro name will be compared to the $LIBx format from each PDS to see if it is a macro in one of the libraries.

The %FINDMACS macro will be called for each macro within each PDS. It parses a single PDS member and writes out one observation for each macro call. If there were no macros called, it writes one observation out to represent the 'calling' macro. The counter ANYMACS is incremented each time a macro call is found; it is renamed in the output data set to represent the called ORDER for the macro within the calling macro. The member-level information is then appended to the LIBx library-level data set. Note that the appropriate library initial is appended to all calling and called macro names.

```
%MACRO FINDMACS(CALLING=,LIBDD=);

  DATA &CALLING(RENAME=(ANYMACS=ORDER
                        ISITMAC=CALLED
                        CALLER=CALLING));
   LENGTH MACLIB $ 40 ISITMAC CALLER $ 11 ;
   RETAIN CALLER "&CALLING (&INITIAL)" ;
   INFILE &LIBDD(&CALLING) END=LASTREC;
   INPUT CODE $ 1-80;
   IF CODE=:'%'   THEN DO;
      ISITMAC = SCAN(CODE,1,'% (');
      MACLIB=LEFT(
         %DO _J = 1 %TO &NUMLIBS-1;
             PUT(ISITMAC,$LIB&&LETTER&_J...)||
         %END;
        PUT(ISITMAC,$LIB&&LETTER&NUMLIBS...));
      IF MACLIB  NE ' ' THEN DO;
         ISITMAC=TRIM(ISITMAC)||' '||MACLIB;
         ANYMACS+1;
         OUTPUT;
      END;
   END;
   IF LASTREC AND NOT ANYMACS THEN DO;
      ISITMAC=' ';
      MACLIB=' ';
      OUTPUT;
   END;
  RUN;

  PROC APPEND BASE=LIB&INITIAL.P
              DATA=&CALLING;
  RUN;

%MEND FINDMACS;
```

The %LOOPFIND macro is executed for each PDS and generates the calls to the %FINDMACS macro for each member in that library. The PDS-level data produced in each call of %LOOPFIND are appended to the data set LEVEL1, which will contain all macros and first-level macro calls.

```
%MACRO LOOPFIND(INITIAL=);

  DATA _NULL_;
   SET LIB&INITIAL END=LASTREC;
   IF START NE 'OTHER' THEN DO;
      CT + 1;
      CALL SYMPUT("VAR"||LEFT(PUT(CT,4.)),
             '%FINDMACS(CALLING='||START||
             ",LIBDD=LIB&INITIAL)");
   END;
   IF LASTREC THEN
      CALL SYMPUT('NUMMACS',PUT(CT,4.));
  RUN;

  %DO _I = 1 %TO &NUMMACS;
      &&VAR&_I;
  %END;

  PROC SORT DATA=LIB&INITIAL.P
              (KEEP=CALLING CALLED ORDER);
   BY CALLING ORDER CALLED;
  RUN;

  PROC APPEND BASE=LEVEL1 DATA=LIB&INITIAL.P;
  RUN;

%MEND LOOPFIND;

%LOOPFIND(INITIAL=&LETTER1)
%LOOPFIND(INITIAL=&LETTER2)
```

The LEVEL1 data set that results from the calls to %LOOPFIND is shown in Figure 3. It contains all the first-level, or direct macro calls within each macro in each PDS as well as the respective order of those calls.

Finding All Nested Relationships

The LEVEL1 data set is the foundation of the processing to expand the nested relationships. This data set will be merged back against itself as many times as necessary to expand all nested macro calls, as illustrated in the Figure 4 representation of the expansion for MACRO4.

Prior to doing the iterative processing needed, the LEVEL1 data set will be transposed to one observation per calling macro. Since the merging will be done by macro names, and since there may be cases where a macro that calls multiple macros is itself called in multiple macros, the transpose is necessary to facilitate the iterative merging without having multiple observations with the same BY-variable values in both data sets. The transposed LEVEL1 data will contain the called macro names in array variables.

```
PROC SORT DATA=LEVEL1;
 BY CALLING ORDER CALLED;
RUN;

DATA LEVEL1_T (KEEP=CALLING NUMCALD
                    CALD1-CALD10);
 ARRAY CALDMACS{10} $ 11 CALD1 - CALD10;
 RETAIN CALD1-CALD10;
 SET LEVEL1;
 BY CALLING ;
 IF CALLED NE ' ' THEN DO;
    NUMCALD + 1;
    CALDMACS(NUMCALD) = CALLED;
 END;
 IF LAST.CALLING  THEN DO;
    OUTPUT;
    IF NUMCALD GT 0 THEN
    DO _I_=1 TO NUMCALD;
       CALDMACS(_I_)=' ';
    END;
    NUMCALD=0;
 END;
RUN;
```

LEVEL1 DATA - 1ST LEVEL RELATIONSHIPS

CALLING	ORDER	CALLED
MACRO1 (A)	1	MACRO3 (A)
MACRO1 (A)	2	MACRO4 (B)
MACRO1 (A)	3	MACRO2 (A)
MACRO2 (A)	1	MACRO4 (B)
MACRO2 (A)	2	MACRO3 (A)
MACRO2 (A)	3	MACRO5 (B)
MACRO3 (A)	0	
MACRO4 (B)	1	MACRO3 (A)
MACRO4 (B)	2	MACRO5 (B)
MACRO5 (B)	1	MACRO3 (A)
MACRO5 (B)	2	MACRO3 (A)

Figure 3

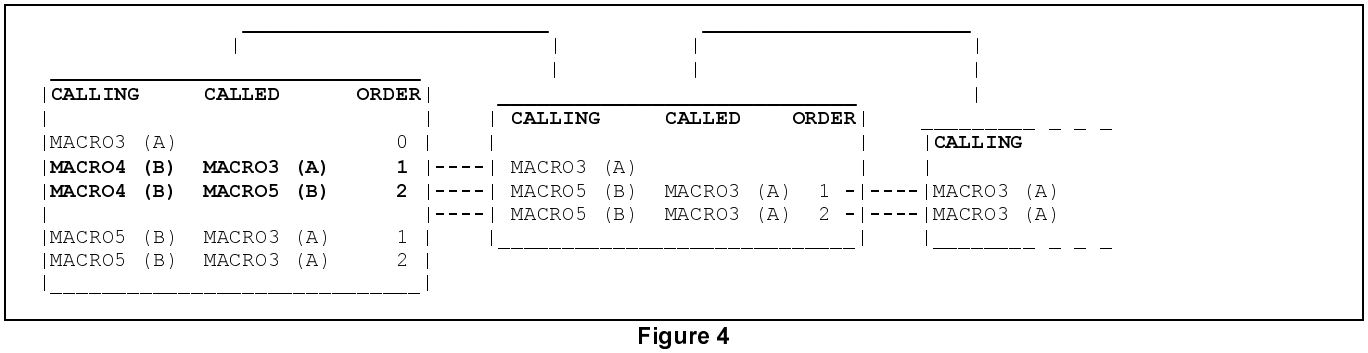

Figure 4

The transposed LEVEL1 data, LEVEL1_T, is shown in Figure 5.

The %MERGMACS macro performs the iterative merging necessary to match all called macros against calling macros until there are no more called macros (&ANYLEFT=0). Starting with LEVEL1, the data set is sorted by the called macro names, and the variables are renamed. The original CALLING macros are renamed to MACLV_1; the order of the called macros is renamed to ORDLV_1; and the called macros are renamed to CALLING. Each iteration of the %DO %WHILE has an associated &LEVLIN and &LEVLOUT, which represent the level of macro calls that exist in the growing expanded calls data set that will be merged with LEVEL1_T to capture the &LEVLOUT level of macro calls. The determination of whether there are &ANYLEFT is made by counting any called macros within the merge.

```
%MACRO MERGMACS;

  %LET ANYLEFT=99;
  %LET LEVLIN=1;
  %LET LEVLOUT=2;
  %DO %WHILE(&ANYLEFT > 0);

      PROC SORT DATA=LEVEL&LEVLIN
                OUT= LEVEL&LEVLOUT
               (RENAME=(CALLING=MACLV_&LEVLIN
                        ORDER =ORDLV_&LEVLIN
                        CALLED=CALLING));
       BY CALLED ;
      RUN;

      DATA LEVEL&LEVLOUT
          (KEEP=MACLV_1-MACLV_&LEVLIN
                ORDLV_1-ORDLV_&LEVLIN
                ORDER CALLING CALLED );
       ARRAY CALDMACS{10} $ 11 CALD1 - CALD10;
       RETAIN TOTCALD 0;
       MERGE LEVEL1_T(IN=CALD)
             LEVEL&LEVLOUT(IN=LEV&LEVLOUT)
             END=LASTREC;
       BY CALLING ;
       IF LEV&LEVLOUT THEN DO;
          IF NUMCALD GT 0 THEN DO;
             DO _I_=1 TO NUMCALD;
                CALLED = CALDMACS(_I_);
                ORDER  = _I_;
                OUTPUT;
             END;
             TOTCALD + 1;
          END;
          ELSE DO;
             CALLED=' ';
             ORDER=.;
             OUTPUT;
          END;
       END;
       IF LASTREC THEN
         CALL SYMPUT
              ('ANYLEFT',PUT(TOTCALD,4.));
      RUN;

     %IF (&ANYLEFT > 0) %THEN %DO;
        %LET LEVLIN=%EVAL(&LEVLIN + 1);
        %LET LEVLOUT=%EVAL(&LEVLOUT + 1);
     %END;

   %END;

   PROC DATASETS LIB=WORK;
    MODIFY LEVEL&LEVLOUT;
    RENAME CALLING=MACLV_&LEVLOUT;
    CHANGE LEVEL&LEVLOUT = ALLCALLS;
   RUN;

%MEND MERGMACS;

%MERGMACS
```

Formatting the Report

Once the macro relationships are determined, there is still the problem of how to display the information. Figure 6 shows a plain listing of the final ALLCALLS data set that is difficult to read because of the repeating values in the nested level columns.

LEVEL1_T DATA (TRANPOSED LEVEL1 DATA)

CALLING	NUMCALD	CALD1	CALD2	CALD3	CALD4	CALD5	CALD6	CALD7
MACRO1 (A)	3	MACRO3 (A)	MACRO4 (B)	MACRO2 (A)				
MACRO2 (A)	3	MACRO4 (B)	MACRO3 (A)	MACRO5 (B)				
MACRO3 (A)	0							
MACRO4 (B)	2	MACRO3 (A)	MACRO5 (B)					
MACRO5 (B)	2	MACRO3 (A)	MACRO3 (A)					

Figure 5

MACLV_1	MACLV_2	MACLV_3	MACLV_4	ORDLV_1	ORDLV_2	ORDLV_3
MACRO2 (A)	MACRO4 (B)	MACRO3 (A)		1	1	.
MACRO2 (A)	MACRO4 (B)	MACRO5 (B)	MACRO3 (A)	1	2	1
MACRO2 (A)	MACRO4 (B)	MACRO5 (B)	MACRO3 (A)	1	2	2
MACRO2 (A)	MACRO3 (A)			2	.	.
MACRO2 (A)	MACRO5 (B)	MACRO3 (A)		3	1	.
MACRO2 (A)	MACRO5 (B)	MACRO3 (A)		3	2	.
MACRO3 (A)				0	.	.
MACRO4 (B)	MACRO3 (A)			1	.	.
MACRO4 (B)	MACRO5 (B)	MACRO3 (A)		2	1	.
MACRO4 (B)	MACRO5 (B)	MACRO3 (A)		2	2	.
MACRO5 (B)	MACRO3 (A)			1	.	.
MACRO5 (B)	MACRO3 (A)			2	.	.

Figure 6

Using a PROC PRINT with the BY variables=ID variables is also a problem because the depth to which the duplicate values need to be blanked out can change with each line. The formatted report for this application is done by pre-processing the fully expanded macro call data set and manually determining where all repeated values are that can be eliminated. To make the listing more readable dashes are used instead of blanks for repeated data.

This macro determines the maximum level of nesting for each record in the data set. Then it uses the values for the called macro order variables (ORDLV_1 for MACLV_2, ORDLV_2 for MACLV_3, etc.) to determine if the macro name at that level represents a new call or a repeat of an earlier one. When a new call occurs, the macro name will be shown. The order variables need to be used rather than just the macro name variables because of the possibility of consecutive calls to the same macro.

```
%MACRO PRTMACS;

  DATA _NULL_;
   RETAIN MAXINDEX 0;
   SET SASHELP.VCOLUMN END=LASTREC;
   IF LIBNAME='WORK' AND
      MEMNAME='ALLCALLS' AND
      NAME=:'MACLV_'
   THEN
      MAXINDEX=
      MAX(MAXINDEX,INPUT(SUBSTR(NAME,7),2.));
   IF LASTREC THEN CALL SYMPUT
         ('MAXINDEX',LEFT(PUT(MAXINDEX,2.)));
  RUN;

  %LET MAXLESS1 = %EVAL(&MAXINDEX - 1);

  DATA PRINT (KEEP=MACLV_1 - MACLV_&MAXINDEX
                   ORDLV_1 - ORDLV_&MAXLESS1
                   MAXLEVEL);
   ARRAY NESTED (&MAXINDEX) $ 11
                 MACLV_1 - MACLV_&MAXINDEX;
   RETAIN MAXLEVEL 0;
   SET ALLCALLS
       END=LASTREC;
   DO I=1 TO &MAXINDEX;
      IF NESTED(I) NE ' ' THEN
         MAXLEVEL=MAX(MAXLEVEL,I);
   END;
  RUN;

  PROC SORT DATA=PRINT;
   BY MACLV_1 ORDLV_1 - ORDLV_&MAXLESS1;
  RUN;

  DATA PRINT;
   ARRAY PRT{*} $ 11 PRT_2 - PRT_&MAXINDEX;
   ARRAY MAC{*} $ 11
                MACLV_2 - MACLV_&MAXINDEX;
   ARRAY ORD{*} ORDLV_1 - ORDLV_&MAXLESS1;
   ARRAY PREVORD{*} PRVORD1 - PRVORD&MAXLESS1;
   RETAIN  PRVORD1 - PRVORD&MAXLESS1 NEWCALL;
   SET PRINT;
   BY MACLV_1 MACLV_2 NOTSORTED;
   IF FIRST.MACLV_2 THEN DO;
      DO _I_ = 1 TO (MAXLEVEL - 1);
         PRT(_I_) = MAC(_I_);
         PREVORD(_I_) = ORD(_I_);
      END;
      OUTPUT;
   END;
   ELSE DO;
      NEWCALL=0;
      DO _I_ = 1 TO (MAXLEVEL - 1);
         IF ORD(_I_) = PREVORD(_I_) AND
            NOT NEWCALL THEN
               PRT(_I_) = '-----------';
         ELSE DO;
            NEWCALL = 1;
            PRT(_I_) = MAC(_I_);
         END;
         PREVORD(_I_) = ORD(_I_);
      END;
      OUTPUT;
   END;
  RUN;

  PROC PRINT U LABEL DATA=PRINT;
   BY MACLV_1;
   ID MACLV_1;
   PAGEBY MACLV_1;
   VAR PRT_2 - PRT_&MAXINDEX;
   LABEL MACLV_1  = 'CALLING MACRO'
     %DO I=2 %TO &MAXINDEX;
        %LET ILESS1=%EVAL(&I - 1);
         PRT_&I  =
            "CALLED LEVEL &ILESS1"
     %END;  ;
   TITLE1
      'FORMATTED LISTING OF ALL MACROS'
      ' CALLED IN THE ORDER CALLED';
   TITLE3 ' ';
   FOOTNOTE1
   'THIS INCLUDES MACROS FROM THE LIBRARIES: '
     %DO I=1 %TO &NUMLIBS;
         %LET FNOTNBR=%EVAL(&I + 1);
         FOOTNOTE&FNOTNBR
   "                 (&&LETTER&I) - &&PDS&I ";
     %END;
  RUN;

%MEND PRTMACS;

%PRTMACS
```

Figure 7 shows the resulting output for example macro MACRO2.

```
                FORMATTED LISTING OF ALL MACROS CALLED IN THE ORDER CALLED

              CALLING        CALLED         CALLED         CALLED         CALLED
               MACRO         LEVEL 1        LEVEL 2        LEVEL 3       LEVEL 4

             MACRO2 (A)    MACRO4 (B)     MACRO3 (A)
                           -----------    MACRO5 (B)     MACRO3 (A)
                           -----------    -----------    MACRO3 (A)
                           MACRO3 (A)
                           MACRO5 (B)     MACRO3 (A)
                           -----------    MACRO3 (A)

                           THIS INCLUDES MACROS FROM THE LIBRARIES:
                                          (A) - USERID.SUGI22.TLIBA
                                          (B) - USERID.SUGI22.TLIBB
```

Figure 7

Conclusion

In practice, this processing is implemented as multiple batch jobs. One job does all the processing up to the %PRTMACS macro, and saves the resulting data set. Ad-hoc report requests are then run against this data set for the reports on one or more macros using the processing in the %PRTMACS macro plus code to subset for the requested macros and to determine the deepest level of nesting present for the requested macros. The report prints only as many levels of nesting as needed. Another report run against the data finds all macros that call a particular macro. This is extremely useful if a low-level macro needs to be modified and the analysts need to assess the impact.

This processing can be used on SAS programs as well as macros. It generates documentation that is extremely useful in analyzing, learning, and documenting existing systems. The documentation is also good for debugging.

Acknowledgements

Access to TSO and to the SAS System for MVS for development of the examples used in this poster was generously provided by Eli Lilly & Company.

The author thanks Martha Thieme of Profound Consulting for editorial review of this poster.

Connie Bryant
Computer Task Group, Inc.
5875 Castle Creek Parkway, Suite 208
Indianapolis, Indiana 46250
(317) 578-5100
cbryant@indy.net