DEVELOPING AND USING MACRO UTILITY LIBRARIES

DAVID S. SEPTOFF, ORI INC.
KATIE A. HUBBEL, ORI INC.

1. INTRODUCTION

This tutorial describes the advantages of using a SAS Macro Utility Library. A Macro Utility Library is a collection of macros which perform functions common to many applications. These macros can easily be used within any program by using the %INCLUDE statement. Given the capabilities of the %INCLUDE statement, one may ask, "Why Macros?", why not just %INCLUDE code to perform the needed task? The answer is simple. The Macro Facility has increased our ability to use SAS as a programming language. The Macro Facility contains features that basic SAS does not. Specifically, Macros can be used to:

- replace repetitive code;
- conditionally execute code;
- execute parameter driven code (this becomes increasingly important when in a system environment);
- retain variables across SAS steps;
- package long, detailed programs making them available with one simple command (the macro call).

What is meant by a Utility Library? The word "Utility" in this sense means a tool that can be used in a variety of situations. The word "Library" means that these Macros are kept in a commonly accessible area. For example, in an OS batch environment, a partitioned data set (PDS) could be used as the library. In a CMS environment, a minidisk could be dedicated to the Utility library (see Figure 1).

To conclude, a SAS Macro Utility Library is a set of Macros stored together in an easily accessible place with each Macro being a tool usable by programmers and users in building and testing SAS programs and systems.

2. Building the Utility Library

What should be included in a Macro Utility Library depends on the needs of those who would use the library. A general rule is that any Macro which performs a commonly needed function should be included. There is virtually no limit on this type of Macro. Some examples would be:

- creation of SAS FORMATS from information stored in SAS data sets (see Figure 1);
- creating a Macro variable whose value is the number of observations in a data set;
- conditionally executing (if there are transactions) an UPDATE step.

Perhaps the most useful types of Macro Utilities are those that facilitate the program development process. These are Macros that assist the programmer in debugging and testing of code. For example, when testing a program, a list of data sets are needed to test whether the logic is flowing properly. Frequencies are also useful as a debugging tool. Both of these debugging tools can be put together as Macros with parameters that will allow the user to suppress the prints or frequencies when they are no longer needed for testing. By changing parameters as opposed to removing code, three important things are accomplished:

- Saving the time of having to reenter the code if more testing is needed.
- Avoiding careless errors that the programmer might make in the process of deleting code.
- Avoiding making major changes in previously tested code. The only change that might be needed is the parameter on the Macro call.

The main objective of using the library is to simplify the life of the programmer or analyst who is running jobs or in the building of a system. The Macro in Figure 2 simplifies the testing process of a system by setting the system options according to the

1141
resolution of &DEBUG. This Macro can be placed at the beginning of a system. It dictates two things to the rest of the system:

- By setting the parameter "DEBUG" to resolve to the character string YES, it signifies to the rest of the system that testing is being done and to turn on the debugging tool Macros. If the run is not a test run, the user can change the parameter "DEBUG" to resolve to anything except for YES and the debugging tool Macros will all be turned off.

- If &DEBUG does not resolve to YES then the source code is turned off. The only output that is seen is output desired by the user and generated by the system (other options can be added). For instance, if the user does not want notes printed in production runs, the NONOTES option should be added next to the NOSOURCE option.

The TSTPRNT and TSTFREQ Macros (shown in Figures 3 and 4) are examples of such Macro tools. One can see that both of these Macros execute only if &DEBUG resolves to YES. To illustrate the advantages of TSTPRNT and TSTFREQ, consider the following example:

```sas
DATA CLASS RANK GRAD;
  %TSTPRNT (IN=CLASS, OBS=20, VARLIST=X Y Z)
  %TSTPRNT (IN=RANK, OBS=50, BYVARS=X Y)
  %TSTPRNT (IN=GRAD, OBS=25, VARLIST=X Y, IDLIST=X)
  %TSTFREQ (IN=CLASS, TABLES=SEX)
```

Here is an example of performing the same function using TSTPRNT and TSTFREQ from the Macro Utility Library:

```sas
%LET DEBUG=YES;
%INCLUDE MACLIB(TSTPRNT,TSTFREQ);
DATA CLASS RANK GRAD;
  ... sas code ...
  ... ;
%TSTPRNT(IN=CLASS, OBS=20, VARLIST=X Y Z)
%TSTPRNT(IN=RANK, OBS=50, BYVARS=X Y)
%TSTPRNT(IN=GRAD, OBS=25, VARLIST=X Y, IDLIST=X)
%TSTFREQ(IN=CLASS, TABLES=SEX)
```

The real advantage of using the library Macros shows up when testing is completed and the code is ready for production. Now all of the prints of the intermediate datasets must be removed from the code. This can be done by simply changing the %LET statement so that the value of &DEBUG is set to NO. Thus all the test output can be turned on and off by changing just one statement, %LET.

A third category of Macros are those that do error checking within a system. The error checking Macros do not check for syntax errors, that is done by the SAS supervisor, these are errors that might, for example, be in the data, or errors that might be in the way the
Jeffrey Varjah

3.1 DISPLAY MACRO STATUS

%Put statements should be used to inform the user on the log when the Macro is loaded (i.e. brought into the job stream). This occurs externally to the Macro and when the Macro is beginning Macro processing (inside the Macro). The format for the %PUT statements are as follows:

%MACRO TSTPRNT();
%PUT NOTE: *** MACRO TSTPRNT IS NOW BEGINNING MACRO PROCESSING;

... SAS code ...

%HEND TSTPRNT;
%PUT NOTE: *** MACRO TSTPRNT IS NOW LOADED;

The purpose of using these %PUT statements is to identify, for the sake of clarity, when the Macros are defined to the SAS job stream (loaded) and when they begin to execute (begin Macro processing). This is a tremendous help when trying to identify where errors are occurring when testing a run using Macros. In addition, if a Macro-based system is being tested, the advantage of this convention becomes even more apparent. Often obscure messages are generated by the Macro Facility when an error is encountered. It is also hard to pinpoint exactly where the error is and which Macro the error might be in. If the first message (MACRO <Macro name> IS NOW LOADED) is generated then the user knows that the Macro was brought into the system and that the error must be further on. The important message to look for is the second one (MACRO <Macro name> IS BEGINNING PROCESSING). This message signifies whether the Macro has begun processing or not.

This is how your log might look if the "NOTE" coding convention was used and an error was encountered. By adding the "NOTE" convention time was saved in trying to determine which Macro the error has shown up in (the error in this case has shown up in Macro TSTP1). The reason for the "NOTE:" is that when looking at the job on the screen and the user asks to see the notes, both the user notes and the SAS generated notes will appear. To distinguish between the two three stars "***" were put within the note.

3. MACRO CODING CONVENTIONS

It is very important that throughout the Macro Utility Library strict coding conventions be followed. This avoids needless errors by users of the library. This also enforces continuity throughout the library. In the following subsections some coding conventions are recommended.


Errors are checked for within the module, using regular SAS code, and Macro variables are set using the Macro function SYMPUT if there are errors within a particular module. These Macro variables direct the execution of the error checking Macro. An example of this type of Macro is seen in Figure 5.

The programmer must decide on the conditions in which the system is not running through correctly. If the error conditions are met then the system should be set to a Macro variable, "_ERROR" for example, to the character string 'YES'. Or the basis of a _ERROR, when the error checking Macro is called, the system either executes the Macro (if the condition is 'YES') or does not. Within this Macro provisions, can be made to stop the system if a STOP is set to 'YES' or to have the Macro print a message on the SAS log, or both, see Figure 6. Because Macros are being used instead of regular SAS code, other Macros may be conditionally executed, for example, the output of an error file (which would be printed by the test print macro) which included all errors in the data if desired.

Another suggestion for this type of Macro is to have one that does a PROC EXPLODE and stops the system when data errors occur. These types of Macros have saved both time and computer resources because, when you are testing a system built with Macros (as in "Designing Macro-Based Systems", Jeffrey Phillips, and Jesse Gary, SUGI, 1985) and there are errors in one of the modules, the error checking Macros can flag the error and stop the system at the end of that module before any more computer time or money is wasted.

A. DISPLAY MACRO STATUS

%Put statements should be used to inform the user on the log when the Macro is loaded (i.e. brought into the job stream). This occurs externally to the Macro and when the Macro is beginning Macro processing (inside the Macro). The format for the %PUT statements are as follows:

%MACRO TSTPRNT();
%PUT NOTE: *** MACRO TSTPRNT IS NOW BEGINNING MACRO PROCESSING;

... SAS code ...

%HEND TSTPRNT;
%PUT NOTE: *** MACRO TSTPRNT IS NOW LOADED;

The purpose of using these %PUT statements is to identify, for the sake of clarity, when the Macros are defined to the SAS job stream (loaded) and when they begin to execute (begin Macro processing). This is a tremendous help when trying to identify where errors are occurring when testing a run using Macros. In addition, if a Macro-based system is being tested, the advantage of this convention becomes even more apparent. Often obscure messages are generated by the Macro Facility when an error is encountered. It is also hard to pinpoint exactly where the error is and which Macro the error might be in. If the first message (MACRO <Macro name> IS NOW LOADED) is generated then the user knows that the Macro was brought into the system and that the error must be further on. The important message to look for is the second one (MACRO <Macro name> IS BEGINNING PROCESSING). This message signifies whether the Macro has begun processing or not.

This is how your log might look if the "NOTE" coding convention was used and an error was encountered. By adding the "NOTE" convention time was saved in trying to determine which Macro the error has shown up in (the error in this case has shown up in Macro TESTP1). The reason for the "NOTE:" is that when looking at the job on the screen and the user asks to see the notes, both the user notes and the SAS generated notes will appear. To distinguish between the two three stars "***" were put within the note.

1143
NOTE: *** NACRO TEST1 IS BEGINNING MACRO PROCESSING

ERROR: some kind of error message

NOTE: *** NACRO TEST2 IS BEGINNING MACRO PROCESSING

3.2 INDENT FOR CLARITY.

%IF &DEBUG=YES %THEN 
  &DO;
  PROC PRINT DATA=CLASS;
  TITLE1 CLASS DATA SET;
  RUN;
%END;

3.3 MACRO COMMENTS

Macro Comments (%*...*:) should be used for a narrative of the Macro's purpose and function; any special information that needs to be included to run the Macro; along with where it is stored and its creation date; and who can be referenced for any questions a user may have about the Macro. This information should appear at the beginning of the macro as shown below.

%***********************************************************************
%* THIS MACRO PERFORMS A PROC *;
%* PRINT. WHEN THE GLOBAL VARIABLE *;
%* "DEBUG" IS SET TO "YES" THEN THIS *;
%* MACRO WILL EXECUTE, OTHERWISE THE *;
%* PARAMETER "REPORT" MUST BE SET TO *;
%* "YES". *
%* STORED: location where stored *
%* UPDATED BY DAVID SEPTOFF *;
%* REFERENCE: DAVID SEPTOFF OR *
%* KATIE BOBBLE *
%***********************************************************************

3.4 NAMING CONVENTIONS

Under an OS BATCH environment, if a partitioned data set is used to store the library, then the PDS member names should be the same as the Macro name. Under CBOS make the Macro name the same as the file name.

3.5 PARAMETER SPECIFICATION

All Macro parameters should be keyword, not positional. With keyword parameters, the mistake of getting the parameters in the wrong order cannot be made. It is easier to add new parameters and default values can be set up in the Macro itself.

To make it easier for users and to ensure that programmers have guidelines, set up standard parameter names for commonly used parameters. Some suggested standards for parameters might be:

IN = for input SAS data set.
INDD = for an input OS data set.
OUT = for output SAS data set.
OUTDD = for output OS data set.
OBS = for number of observations.
VARLIST = Variables in a VAR list.
BYVARS = Variables in a BY list.

PL/I comments (%*...*:) should be used for describing variables within the code. In addition, PL/I comments should be used after each parameter within the %MACRO statement to describe the use of that particular parameter. The commenting is done as follows:

%MACRO TSTPRNT(pl, p2=);/* describe pl */
  p2= /* describe p2 */
%END;

4. CONTROLLING THE LIBRARY

After storing the library, set up an organized way of updating the library. Have a committee set up that meets on a regular basis to make these decisions. Restrict write access only to members of the committee or to a subset of the members to insure thorough testing of the Macros that go into the library and to insure the security of the library itself. Have the committee regularly inform other staff members on the updating of the library.

The committee should keep a table of contents of what is in the library. This table of contents should be the first thing someone sees if they do a PROC SOURCE and should contain the block Comments from all of the Macros within the library.
5. CONCLUSIONS

A SAS Macro Utility Library can be a powerful tool in the building and testing of large Macro based systems and small SAS jobs.

6. ACKNOWLEDGEMENTS

The authors would like to thank all members of the CASD/SATPO division of ORI, INC. for their patience throughout this project. In particular, Don Henderson and Jeffrey Phillips for their invaluable advice and Kris Eaynes for her outstanding job in typing this manuscript.

The authors will respond to all questions and suggestions at:

ORI, INC.
122 C STREET N.W. SUITE 250
WASHINGTON D.C. 20001
(202) 737-2666

7. REFERENCES

The authors would like to thank all members of the CASD/SATPO division of ORI, INC. for their patience throughout this project. In particular, Don Henderson and Jeffrey Phillips for their invaluable advice and Kris Eaynes for her outstanding job in typing this manuscript.

2. PHILLIPS, JEFFERY and GARY, JESSE, DESIGNING MACRO BASED SYSTEMS, SUGI, 1985.
3. SAS USER'S GUIDE: BASICS, 1982 EDITION.

FIGURES

FIGURE 1 - SIMPLE SAS FORMAT BUILDING MACRO

%MACRO MAKFMT(FROMDATA=, KEY= RESULT=, NAME=)

DATA _NULL_;
  SET &FROMDATA END=LASTREC;
  @VAR='_'._FMT| 'LEFT(PUT(_N_,4.));
  CALL SYMPUT('VAR','^' || TRIM(&KEY) || '=' || TRIM(&RESULT) || ' ');
  IF LASTREC THEN CALL SYMPUT('ENTRIES',PUT(_N_,4.));
RUN;

PROC FORMAT;
  VALUE &NAME
    &DO 1 = 1 TO &ENTRIES;
    &_FMT&&
  &END;
%
%MEND MAKFMT
FIGURE 2 - SYSTEM DEBUGGING TOOL

%MACRO TSTOPT(DEBUG=);
    %IF &DEBUG = YES %THEN
        %LET OPTTEST = sas debugging options;
    %ELSE
        %LET OPTTEST = NOSOURCE;
    OPTIONS &OPTTEST;
%MEND TSTOPT;

FIGURE 3 - TEST PRINT MACRO

%MACRO TSTPRINT(IN=_LAST_, BYVARS=, VARLIST=, PROPTS=,
                 IDLIST=, TITLE=, OBS=500, TITELE=, TITELEB=)

%LOCAL DSN;
%IF &DEBUG=YES %THEN
    %DO;
        %IF &IN=_LAST_ %THEN
            %LET DSN=%SCAN(&SYSDSN,1) . %SCAN(&SYSDSN,2)
        %ELSE %LET DSN=&IN;
        PROC PRINT DATA = &DSN(OBS=&OBS) &PROPTS;
        VAR &VARLIST;
        BY &BYVARS;
        ID &IDLIST;
        %UNQUOTE(TITLE=EVAL(&TITLE+1)) &TITLE;
        %UNQUOTE(TITLE=EVAL(&TITLE+2)) &TITLE;
        RUN;
    %END;
%MEND TSTPRINT;
%MACRO TSTFREQ(IN=, TABLEST=, TLINE=);
  %IF &DEBUG = YES %THEN
    %DO;
      PROC FREQ DATA = &IN;
      TABLES &TABLEST;
      %UNQUOTE(TITLE=EVAL(&TLINE+1)) FREQUENCY OF DATA SET &IN;
      RUN
    %END;
  %END TSTFREQ;

%MACRO ERRSET(MODULE=, ERMSG=, STOP=);
  %IF &_ERROR = YES %THEN
    %DO;
      %IF &STOP = YES %THEN
        %DO;
          %PUT MODULE &MODULE HAS ENDED;
          %PUT &ERMSG;
          ENDSAS;
        %END;
      %ELSE
        %PUT &ERMSG;
    %END;
  %END ERRSET;
FIGURE 6 - USING THE ERROR CHECKING MACRO

```
IF PRICE < 0 THEN
  CALL SYMPUT('_ERROR','YES');

RUN;
%ERRSET(MODULE=SETPRICE, ERRMSG=INVALID PRICE, STOP=YES)
```