DEVELOPING A SAS SYSTEM AUTOCALL MACRO LIBRARY AS AN EFFECTIVE TOOLKIT

Steven A. Wilson, Kaiser Permanente Division of Research, Oakland, CA.

ABSTRACT

This paper discusses using the SAS System Autocall Facility for the development of a common set of macro utilities for use by application programmers. The autocall facility allows users to invoke a SAS macro without having to define the macro in their current job or session. This facilitates the development of a centralized "toolbox" of SAS macros that many individuals may utilize. Additionally, the development of an effective autocall library within a systems group allows programmers to quickly perform common processing tasks with consistency and accuracy.

Note that this is not a presentation on macro coding. Instead, the focus of this paper is using SAS macros as utilities. Techniques for developing and managing an effective macro library are discussed.

Introduction - A simple macro called NOBS

To illustrate the topics presented, a simple macro will be developed and referred to throughout this paper. This simple macro creates a macro variable that resolves to the number of observations contained in a passed SAS data set. The macro code follows:

```
%MACRO NOBS(DSN);
  %GLOBAL NOBS;
  DATA _NULL_;
    SET &DSN NOBS=COUNT;
    CALL SYMPUT('NOBS',COUNT);
    STOP;
  RUN;
%MEND NOBS;
```

The %GLOBAL statement causes the macro processor to create the macro variable NOBS in the global referencing environment. Thus, the macro variable NOBS will exist for the duration of the SAS session.

The NOBS= option on the SET statement specifies a variable to be created whose value will be the number of observations contained in the specified data set.

The CALL SYMPUT routine simply passes the value of the DATA step variable COUNT into the macro variable NOBS.

One use for this macro follows:

```
%nobs(PRINT);

PROC PRINT DATA=PRINT(OBS=25);
TITLE 'DATA PRINT (First 25 observations)';
FOOTNOTE "&nobs obs are in the data set";
RUN;
```

Setting Up and Using a SAS Autocall Library

The SAS AUTOCALL facility allows you to store SAS macros in external files and invoke them without having to define or include them in your current program or session. For example, by storing the macro NOBS in an autocall library, many jobs may access this macro, eliminating the need to program the same code in each job.

One benefit of using the Autocall facility is that macro code is stored in only one place. Thus, enhancements to a macro need be made in only one place instead of numerous jobs. This allows you to accumulate useful macro utilities for easy access and maintenance.

The following four easy-to-complete steps show how to create and use a SAS autocall macro library:

Step 1: Allocate a SAS Autocall Library

Create a library to store source code for macros.

Depending upon your operating system, different host system data sets are allocated for this purpose.

Platform	Allocate the following
UNIX & PC	Directory
MVS	Partitioned Data Set (PDS)
CMS	Maclib

Step 2: Insert Macros into the Autocall Library

Store each macro as a separate PDS member or file in a directory.

The name of the PDS member or file that contains the stored macro source code must be the same name as the macro defined by the code. For

example, only the source code for macro NOBS should be placed in the file named NOBS.

Only store one macro in each PDS member/file.

Your operating system may have some rules of syntax for naming these PDS members or files. Syntax issues regarding site specifics should be referred to the SAS Companion manual for your platform.

A couple of examples:

* The file extension in UNIX and PC must be .SAS.

* In MVS, use a pound sign (#) in the PDS member name in place of any underscore (_) in the macro name. MVS does not allow the underscore character as part of a PDS member name. For example, put the SAS macro %N_OBS into a PDS member named N#OBS.

Step 3: Assign the Autocall Library in a SAS session
Use the SAS AUTOCALL feature to optimally allocate the previously created macro library. To do this allocation, two SAS options must be specified:

MAUTOSOURCE
This SAS option simply enables the autocall facility. When MAUTOSOURCE is in effect and a macro call occurs for a macro that has not yet been encountered, the macro processor will search the autocall library for a file with the same name as the called macro.

SASAUTOS = file-specification
This SAS option specifies the file(s) that contain your autocall macros. Again, there are host specific syntax rules that must be adhered to when specifying this option at SAS invocation or in a configuration file. However, when using this option in an OPTIONS statement, the syntax is the same across all platforms.

Multiple macro libraries may be concatenated together to form the Autocall library that SAS will use in a session.

```
SASAUTOS =
      (file-spec-1, file-spec-2, ...)
```

When this occurs, SAS will search the macro libraries in the order listed until the matching PDS member or file name is encountered.

The following box shows an OPTIONS statement for assigning an autocall library in a SAS session:

```
OPTIONS MAUTOSOURCE
        SASAUTOS=('MY.SAS.MACRO.LIB',
                  SASAUTOS);
```

The SASAUTOS fileref, by default, points to a macro library that is supplied by the SAS Institute. This library contains many SAS autocall macros supplied by the SAS Institute. Among these autocall macros are the following:

%cmpres	%left	%trim	%datatyp
%verify	%qcmpres	%qleft	%qtrim

These Institute supplied SAS macros are always automatically attached to a SAS session as an autocall library with the fileref of SASAUTOS. Be sure that you do not accidentally de-allocate this SAS Institute autocall library.

The OPTIONS statement shown in the box above specifies that the passed macro library MY.SAS.MACRO.LIB will be searched for the macro source code before the SAS supplied macro library is searched.

Note: Another option here is to create your own SAS exec by copying your site's SAS invocation exec and modifying it to automatically include your library as an autocall library.

Step 4: Call any needed macros in your session
The autocall facility will search the specified autocall library(ies) for a file or PDS member name that matches a called macro. When found, the code contained in the found file or PDS member will be included, compiled, and the macro will then execute. There is no need to use an INCLUDE statement to access the desired macro code.

As when any macro is called, the compiled macro code is stored in the WORK library in a catalog named SASMACR prior to execution (WORK.SASMACR.macname.MACRO). Thus, any subsequent calls to the macro will result in immediate execution of the compiled code. SAS compiles the macro code only once.

With SAS Version 6.07 it is possible to save the compiled macro code that normally is written to the WORK.SASMACR catalog. The Compiled Stored Macro Facility is briefly discussed at the end of this paper.

Creating a Common SAS Macro Library

A macro autocall library to be used by many people should meet certain criteria to be useful. An effective SAS autocall macro library would have each of the following three elements:

MACRO SOURCE CODE
MACRO DOCUMENTATION
MACRO VALIDATION/TUTORING JOB

Depending upon your particular platform, separate libraries or specific file suffixes may be best for containing these different elements. Regardless, a skeleton of each element should be present to insure consistency. Thus, whenever a new file is being created, the skeleton can be copied, providing a basic outline from which to begin.

The next few pages of this paper discuss these elements of a useful common macro autocall library.

MACRO SOURCE CODE LIBRARY

This library contains the SAS macros. Since this library contains only source code, it is not recommended reading for users. Usage of a macro should be determined from the macro documentation and the validation job. Of course the macro source code should be well commented, though.

It is a good idea to restrict access to the source code library so no unauthorized changes may be made to any macro. Since consistency is a desired attribute of any common autocall macro library, it is vital to eliminate the temptation that a programmer may have to tinker with the autocall macros.

Coding conventions should also be developed so the macros will have the same "look & feel" and meet a basic set of user expectations. Following coding standards will also make your macros easier to enhance and maintain.

Some examples of coding conventions follow:

Be Responsive to User's Needs

Create macros that are widely useful and provide enough flexibility to meet the varying needs of users. Also be able to quickly respond to user's questions and solicit suggestions for enhancements.

Design Modular Macros

A macro should be used to perform a single, discrete task. This keeps the number of lines of code in a macro at a minimum and aids in maintenance and de-bugging.

Design Self Contained Macros

A user should be able to call a macro without having to provide the macro with certain variables, files, or other elements. External files needed by the macro should be allocated by the macro.

Design Robust Macros

It is important that common macro utilities be designed in such a way as to prevent abends whenever possible. Macro utilities that frequently abend when being used by inexperienced programmers are a great source of frustration. For example, if a SAS data set name is being passed into a macro, will the macro still operate properly if the passed data set name has any data set options attached to it (such as a WHERE= data set option) or if the data set contains 0 observations.

Keep Macro Code in Lower Case

All macro code should be lower case. This aids in differentiating between SAS statements and macro code when looking at a macro.

Keep Macros Easy to Read

- Code only one statement per line.
- Use lots of blank lines.
- End every DATA and PROC step with a RUN.
- Follow a consistent indentation scheme.
- Use lots of source code comments

First and Last Statements

The %macro statement should be the first statement in the autocall file, while the %mend statement should be the final statement in the file.

Although other statements may appear before or after the autocall macro, these extra statements will only be included when the autocall file is first referenced and not on subsequent calls to the macro.

Use a Descriptive Header in the Macro

Immediately following the %macro statement should be a comment box to briefly describe the macro's purpose. Keep it short since usage documentation should be contained in the macro documentation file. This area may refer the user to the documentation file.

Use Keyword Parameters

Macros should utilize keyword parameters. This not only makes the macro call easier to interpret, but also makes the macro more user friendly. The exceptions are when a macro has only one or two parameters and those parameters are intuitively apparent.

Do Not Overparameterize
Too many parameters in a macro makes it difficult to use and maintain. Accurate comprehension of the actions to be performed by a macro call utilizing many parameters is also difficult.

Make Parameters Easy
Make any YES/NO type parameters easy to use by accepting the following: YES NO Y N yes no y n This can be done by using the following macro code to set these parameters:

```
%let y_n=%upcase(%substr(&y_n,1,1));
```

Check Passed Parameters
Perform edit checking on the passed parameters. If the macro is improperly called, alert the user and abort processing . This prevents wasting CPU time and incurring unnecessary job costs.

```
%if %length(&required) = 0 %then %do;
  %put  Parameter REQUIRED missing;
  /* label %quit: at end of macro */
  %goto quit;
%end;
```

Supply Default Parameter Values
Supply appropriate default values to non-required parameters. This should be done regardless of whether there are default values specified on the %macro statement because the user may have accidentally overwritten the default with a null value.

```
%if %length(&test)=0 %then %let test=N;
```

Supply a TEST Parameter
A parameter named TEST should be present in all macros. This parameter can be turned on to obtain intermediate output and/or other information that can be helpful for de-bugging. The macro can also set MPRINT when in test mode.

```
%if &test = Y %then %do;
   %put  A MESSAGE for DE-BUGGING;
   OPTIONS MPRINT;
   TITLE3 'Intermediate data set';
   PROC PRINT DATA=TEMP;
   RUN;
   TITLE3;
%end;
```

Define Macro Variables
Explicitly define the referencing environment of any created macro variables using a %local or %global statement.

Avoid Hard Coding within Macro
For ease of maintenance, do not hard code libraries or data set names in the macro. Use parameters or a %let at the beginning of the macro to specify these values. This makes maintenance easier when a change occurs to these hard coded values.

Write a Message to the SASLOG
For some macros, it may be useful to have the macro write an extended message to the SASLOG that states what type of processing the macro will perform. This is especially useful for complex macros with various processing options, macros with long execution times, and in cases when calling macros with NOMPRINT or NONOTES.

```
%put  ------------------------------;
%put  MACRO _____ has been invoked.   ;
%put  As called, this macro will....  ;
%put  ------------------------------;
```

Determine How SAS is Executing
Some macros may behave differently when run in batch versus interactive. Use the automatic SAS macro variable &sysenv to determine how the macro is being run.

```
/* Interactive */
%if &sysenv = FORE %then
      STOP;
/* Batch  */
%else %if &sysenv = BACK %then
      ABORT ABEND 99;;
```

Note that there are two semicolons used on the last statement in the above example. The first is used to terminate the %else macro statement, while the other is a trailing semicolon that will end whichever SAS statement is resolved.

Determine Which Version of SAS is Executing
Some macros may need to execute different code depending upon which version of SAS they are running. Use the automatic SAS macro variable &sysver to determine the version of SAS that is executing.

```
/* Version 6.04 executing */
%if &sysver = 6.04 %then ...  ;
```

Titles and Footnotes
Reserve the first 2 title and footnote lines for the user, allowing the macro to use the remaining title and footnote lines. Be sure to remove any titles or footnotes set by the macro when the macro is finished.

Avoid Conflicts with User Assigned Names

If temporary DATA step variables or temporary SAS data sets are created, use an underscore (_) as the first character of the variable or data set name. This is to prevent the created variable or data set name from conflicting with names assigned by the user.

Be Neat and Clean

Explicitly define the attributes of all created DATA step variables that are created by the macro. Also, be sure the macro eliminates any created temporary variables or data sets when finished.

Validate Macros

Re-validate any macro after making enhancements. Enhancements and re-validation should occur in a separate, development library to allow other users to access the original macro until enhancements and re-validation are complete.

An Example - NOBS for many users

Following our coding conventions, our sample macro NOBS could appear in an autocall library being used by many users as illustrated in the next column.

Note that while our coding conventions have made our macro more difficult to read, the macro is now much more robust and unlikely to abend.

MACRO DOCUMENTATION LIBRARY

The second important element of a common autocall library is the macro documentation library. This library consists of uniform documentation forms for each macro in the autocall macro library. These forms describe the macro actions and usage.

Since proper documentation removes the need to look at macro code, this should be a user's starting point in learning how to use an autocall macro. For example, since the SAS macro %trim is documented by the SAS Institute, there is no need to ever look at the actual macro source code that defines %trim.

Additional information for use by developers when enhancing the macro is also useful to include in the macro documentation. Names of reference persons can also be placed on these documentation forms.

The important concept is getting useful information to the user. An un-documented macro is essentially useless in a common macro library.

The documentation form for our sample macro NOBS appears on the next page.

```
%macro nobs(dsn,test=N);
%-----------------------------------;
%* This macro returns the number of
%* observations in a SAS data set as
%* the global macro variable &NOBS
%*
%* Documentation stored as
%*  'COMMON.SAS.MACRO.DOC(NOBS)'
%*-----------------------------------;

/*---------------------------------*/
/*  Define and initialize NOBS.    */
/*---------------------------------*/
%global nobs;
%let nobs=0;

/*-------------------------------*/
/*  Perform parameter checking  */
/*-------------------------------*/
%if %length(&dsn) = 0 %then %let
   dsn = _LAST_;
%if %length(&test) = 0 %then %let
    test = N;
%else %let test =
          %upcase(%substr(&test,1,1));

/*-----------------------------------*/
/*  Set option to prevent abend when */
/*  passed a non-existing data set   */
/*-----------------------------------*/
OPTIONS NODSNFERR;

/*-----------------------------------*/
/* Optimize DATA step, tidy macro var */
/*-----------------------------------*/
DATA _NULL_;
  IF 0 THEN SET &dsn NOBS=NOBS;
 CALL SYMPUT('nobs',
       TRIM(LEFT(PUT(NOBS,8.))));
 STOP;
RUN;
/*-----------------------*/
/* Return to SAS default. */
/*-----------------------*/
OPTIONS DSNFERR;

%if &test = Y or &sysenv = FORE %then
  %put There are &nobs obs. in &dsn;

%mend nobs;
```

Macro Documentation Form

%macro nobs(dsn,test=N);

Library: DOR.COMMON.SAS.MACROS

Programmer: S. Wilson
Validator : D. Peters

Description & Usage

This macro utility is a SAS data step that places the number of observations contained in the specified SAS data set in the global macro variable &nobs.

NOTES: The returned value for NOBS is 0 for non-existent data sets.
Can only be used with Direct Access data sets

Placement

After any DATA or PROC step after the referenced data set (&dsn) has been created.

Parameters (R indicates REQUIRED)

Positional

DSN — Name of the direct access SAS data set that you want the number of observations counted.

Keyword

TEST = N — For testing purposes. Writes &nobs to SASLOG.

DEVELOPER NOTES

REFERENCED MACRO VARIABLES

Local Macro Variables

--NONE--

Global Macro Variables

NOBS — Number of observations in passed data set.

SYSVER — Automatic SAS system macro variable used to determine which version of SAS is running

SYSENV — Automatic SAS system macro variable used to determine how SAS is executing

OTHER REFERENCED MACROS, FILES, or LIBRARIES

--NONE--

MACRO VALIDATION/TUTORING JOB LIBRARY

Macro validation jobs are used to test the macro thoroughly for accuracy in normal and extreme conditions. These jobs should be self-contained and ready to run.

Ideally, a validation job for a macro would be written by someone other than the programmer who coded the macro. This gives a second opinion on the ease of use and comprehension of the documentation.

A validation checklist can also be used to verify that the macro conforms to your department's unique but, of course, stringent standards.

In most cases, testing data can be created within the validation job. When necessary, sample data sets can be placed in a common macro validation library for the validation job to utilize.

The validation program can also be used as a training device to show functionality and use of a macro to new users. Thus, any user who wants to learn a macro's function can simply execute the validation job for that macro.

The validation job for our sample macro NOBS appears in the box on the next column.

```
*************************************;
**    Validation job for NOBS      **;
*************************************;
**---Create testing data sets---**;
DATA A B EMPTY;
  DO I = 1 TO 10;
   DO J = 1 TO 4;
      OUTPUT B;
      IF J = 1 THEN OUTPUT A;
    END;
 END;
RUN;

**------------TEST # 1---------**;
%nobs(A,test=Y);
%put TEST #1 NOBS=&nobs
          Correct NOBS = 10;

**------------TEST # 2---------**;
%nobs(B);
%put TEST #2 NOBS=&nobs
          Correct NOBS = 40;

**------------TEST # 3---------**;
%nobs(EMPTY);
%put TEST #3 NOBS=&nobs
          Correct NOBS = 0;

**------------TEST # 4---------**;
%nobs(NONEXIST)
%put TEST #4 NOBS=&nobs
          Correct NOBS = 0;
```

Stored Complied Macro Facility

With SAS Version 6.07, there is a new facility that allows the storage of macros as compiled code. This increases performance over an autocall macro library because the macro source code does not need to included from an autocall file and compiled.

As mentioned earlier, when SAS compiles a macro, the compiled code is stored in the WORK.SASMACR catalog by default. The stored compiled macro facility gives you the ability to place the SASMACR catalog into a library other than the temporary WORK library, thus saving the compiled code for later use.

To use the stored compiled macro facility, two SAS System options must be specified:

`MSTORED`
This option simply tells SAS that you wish to use the stored compiled macro facility either to store compiled macros or to use previously stored compiled macros.

`SASMSTORE = libref`
This option is used to specify the libref of the SAS library that contains the SASMACR catalog of stored compiled macros. If the SASMACR catalog does not yet exist and macros are compiled, the catalog will be automatically created.

It is important to note that only one stored compiled macro catalog may be used at any one time in a SAS session. Like format catalogs, stored compiled macro catalogs cannot be concatenated. This is the major limitation of the stored compiled macro facility for it prevents using stored compiled macros easily in a development environment.

Once these options have been specified, a macro may be compiled and stored in the requested SAS library by using the `STORE` option of the %MACRO statement. The following box shows an example.

```
/*-------------------------------*/
/* Assign libname to SAS library */
/* containing SASMACR catalog.   */
/*-------------------------------*/
LIBNAME COMPMACR
          'C:\MY\COMPILED\MACROS';

/*-------------------------------*/
/* Activate the stored compiled  */
/* macro facility and reference  */
/* the appropriate SAS library.  */
/*-------------------------------*/
OPTIONS  MSTORED
         SASMSTORE=COMPMACR;

/*-------------------------------*/
/* Compile and store the macro.  */
/*-------------------------------*/
%macro  nobs / store;
  .....NOBS source code.....
%mend nobs;
```

This facility should be used only with macros that are thoroughly de-bugged and in "production". Also, it is important to keep the uncompiled macro source code (preferably in an autocall library) because source code cannot be retrieved from the stored compiled macro code.

When both a stored compiled macro library and an autocall library are allocated in a SAS session, SAS will look in the SASMACR catalog for a stored compiled macro first. If the macro is not found in the SASMACR catalog then SAS will proceed to examine the autocall library for the macro source code.

See O'Connor (1992) for a more complete overview of the stored compiled macro facility.

Conclusion

As shown in this paper, the SAS Autocall facility is amazingly easy to implement and use.

By using the SAS Autocall facility, a common set of macro utilities can be developed to aid programmers perform routine tasks. Having a set of SAS macro utilities is a proven time saver that I have seen dramatically improve programmer productivity over and over again.

Using macro utilities allows programmers to avoid common repetitive programming tasks and concentrate on the unique aspects of a job at hand.

Programmer time is saved because programmers no longer have to "re-invent the wheel" by developing their own code when another programmer may already have coded the exact same task.

Macro utilities also assure consistent results that are often not obtained when multiple programmers develop their own code to perform the same task.

Overcoming programmers reluctance to use a "black box" utility is the main obstacle to having a SAS macro library be effective. To address the user's concerns, it is important that the macros be well written to avoid unseemly abends and also be thoroughly documented and validated.

Finally, it should be noted that effective, flexible macros do take considerably more time to code, document, and test. Given the benefits of a common utility library, though, the extra time spent developing a utility is well worth the effort

References

O'Connor, Susan M. (1992), "Macros Invocation Hierarchy: Session Compiled, Autocall, and Compiled Stored Macros", *Proceedings of the Seventeenth Annual SAS Users Group International Conference*, 19-23

Acknowledgements

Thanks to Dr. Tom Hoffman and Amy Lin of Lederle Laboratories in Pearl River, NY for mentoring me in the wonderful world of SAS macros and having introduced me to the concepts presented in this paper.

Author's Address

Steve Wilson
Kaiser Permanente, Division of Research
3505 Broadway
Oakland, California 94611-5400
(510) 450-2211

INTERNET: saw@dor.kaiser.org

Any comments questions, or details of your experiences are most welcome.

SAS is a registered trademark of the SAS Institute, Inc., Cary, N.C.

UNIX is a trademark of AT&T.