AP10

A Practical Approach to the Stored Compiled Macro Facility in a Clinical Trial Environment

Mirjana Stojanovic, Dorothy Watson, and Donna Hollis
Cancer Center Biostatistics, Duke University Medical Center, Durham, NC

ABSTRACT

This paper will demonstrate the stored compiled macro facility in the multi-user clinical data management environment of Cancer and Leukemia Group B (CALGB). CALGB is a large cancer research cooperative group. Clinical data is gathered from hundreds of institutions and sent to the CALGB statistical center, where it is stored in a single database and processed by dozens of statisticians, programmers and data managers. At any given time CALGB coordinates more than 80 actively accruing clinical trials. This setting necessitates the need for streamlining data cleaning, data queries, and reports.

The compiled stored macro facility permits all production level macros to be stored in one common location. In the multi-user setting it is crucial that all changes to the stored compiled macros in the macro catalog are authorized. The stored compiled macro facility permits a small group of users to control the content of the macro library, while a larger group of users has only read access.

Step-by-step instructions on how to create and use compiled stored macros will be presented. Several examples from the CALGB statistical center will be used to demonstrate the flexibility and efficiency of the facility including multilevel macro programs which allow for easy customization of stored programs.

INTRODUCTION

WHAT IS THE STORED COMPILED MACRO FACILITY?

The stored compiled macro facility (SCMF) is a repository for complied macros. Source code for the macros can be stored in various locations separate from the complied macros. The source code needs to be complied only once and then the complied version can be permanently stored in the SCMF. This differs from an autocall library or other macros, which must be complied during every SAS session. Permanent compilation is a feature of the stored compiled macro facility. Programs in the SCMF can be run and viewed by multiple users, even if they users have read only access.

Creating a permanent SASMACR catalog saves time in daily production. The compiled macro is in one location, while the source code can be stored anywhere. To ease maintenance, storing source code in one central location is recommended. The source code can be saved in the same directory as compiled macros or in another directory if complete control of the source code is required. The compiled macro and source code are different file types to SAS® so having them in the same directory is not confusing.

The name of library where the compiled macros will be stored is a SAS® catalog name SASMACR. Compilation happens once and when the macro is called later, the macro processor executes the compiled macro instead of source code. The creation of this library needs some additional work before everything is setup, but later it can save a lot of programming effort and disk space. It is highly recommended that macros be thoroughly tested before compiling to prevent later corrections. Usually, one person should be responsible for the maintenance of the Macro Library. This does not mean that same person makes all of the changes in all of the stored macros, but it is recommended one person supervise all changes to a macro library (“gatekeeper”). The biggest advantage of this is that the updated and tested macro, after changes are made, is compiled only once and stored permanently. All other users can now call the macro and use it. The possibility of making clones, secret modifications and duplications are not feasible.

ADVANTAGES AND DISADVANTAGES OF THE STORED COMPILED MACRO FACILITY

Advantages:

- No repeated compiling of macro programs that are repeatedly used
- Possibility of displaying the entries in a catalog containing compiled macros saving macro compiling time
- There is no need to save and maintain the source for the macro definition in a different location
- Keeping track of macros is easy
- Storing more then one macro per file
- Compile and Store is faster because there is a decrease in time for searching, %including, compiling and storing in the WORK.SASMACR catalog.

Disadvantage:

- Cannot be moved directly to other operating systems.
- Must be saved and recompiled under new OS at any new location.

When a compiled macro is called in a SAS® program, the macro processor skips the compiling step, retrieves the compiled macro code, and executes the already compiled code. The compiled macro code will not be deleted when the SAS® sessions ends and it can be reused at any time. If any change is made to the macro source code, it must be also updated in the SASMACR catalog as well.

CREATING A STORED COMPLIED MACRO FACILITY (SCMF)

The following sample code illustrates using this simple get_table macro to download the data from an ODBC data source into a permanent reference library.

```
libname db "H:\CALGB\GI\80303\test";
libname mydb odbc prompt ;

%macro get_table(table, out, variable_name, variable_value);

data &out;
  set mydb.&table;
  where &variable_name=&variable_value;
run;

%mend get_table;

 %get_table (table=death, out=db.death, variable_name=study,
              variable_value=90206);

libname mydb clear ;
```

To convert this tested macro to a stored compiled macro the following code is run:

```
libname mydb odbc dsn=CALP prompt ;
options MSTORED SASMSTORE = macrolib;

libname macrolib "N:\SAS\programs\macro_library\";

%macro get_table(table, out, variable_name, variable_value)/STORE SOURCE
                                                    DES="Get a DB table";

data &out;
  set mydb.&table;
  where &variable_name=&variable_value;
run;

%mend get_table;
```

To use the stored macro, assuming the OPTIONS and the macrolib libname statements above have been inserted into your SAS® autoexec.sas file (see below), run:

```
libname db "H:\CALGB\GI\80303\test";
libname mydb odbc prompt ;

 %get_table (table=death, out=db.death, variable_name=study,
                                                    variable_value=90206);
libname mydb clear ;
```

STEPS TO CREATE PERMANENT SASMACR CATALOG

Looking at the examples above:

1. Activate the SYSTEM options MSTORED SASMSTORE = macro_library

2. **MSTORED**=Allows the SAS® System to search for compiled macros stored in the SASMACR catalog. It is a system option which enables storage of compiled macros in a permanent SAS® library.

   SASMSTORE= Specify the libname that stores the compiled macros in the catalog SASMACR . Libname cannot be WORK. It is a system options that designates a permanent library to store compiled macros.

3. Create the libname with the file reference where you want to store the macro.

   Libname macrolib "N:\SAS programs\macro_library\".

   General form of a Macro definition for permanent macro storage:

```
%macro macro_name (parameters)  / STORE SOURCE DES="description";
          .
          .
    macro program code
          .
          .
%mend macro_name;
```

When the macro processor finds the macro program in SASMACR catalog, it submits it for immediate execution (compilation was already done before).

There are a few important options to note in the macro statement. These options are:

- **STORE** option = stores the compiled macro in a SAS® catalog in a permanent SAS® data library.
- **SOURCE** option = stores the macro source code along with the compiled code (new in SAS® version 9.1). The SOURCE option requires that the STORE option and MSTORED option be set. The code saved by SOURCE option begins with the %MACRO and ends with %MEND statement. If this option is not selected, other macro users will not be able to access the macro code.
- **DES** option = specifies a description for the macro entry in the macro catalog. It must be in quotation marks. The description appears in the CATALOG window when you display the contents of the catalog containing the stored compiled macro facility.

MULTIPLE USERS UTILIZING THE STORED COMPLIED MACRO FACILITY

Once tested macros have been added to the stored compiled macro facility, they can be run by many users with or without granting access to the source code. This can be done by requiring all users to add the macro library reference to their autoexec.sas file, and then giving them documentation on how to execute each macro. Once the following code is added to a users autoexec.sas file, all stored compiled macros in the SAS® catalog named SASMACR located in the folder 'N:\SAS programs\macro_library'.

```
libname macrolib  ' N:\SAS programs\macro_library ';

options mstored sasmstore=macrolib;
```

HOW DO YOU FIND INFORMATION ABOUT AND WHICH MACROS ARE STORED IN A CATALOG?

To display information about compiled macros when you invoke them, use the SAS® system options, MLOGIC, MPRINT, and SYMBOLGEN. When you specify the SAS® system options MLOGIC, the libref and date of compilation of a stored compiled macro are written to the log.

To display a listing of the macros available in a SASMACR catalog, use PROC CATALOG as follows:

```
libname macrolib 'N:\SAS\Programs\macro_library';

PROC CATALOG catalog=macrolib.sasmacr;
    contents;

Quit; * <=== Note: important to include or proc continues to run;
```

It is not uncommon for people to ask to see the source code once the macro code is stored in a SAS® macro catalog. Users want to know if they will obtain the proper results. Did the macro work in the way that it was intended, etc.? In other words, they want to see and retrieve the source code. With SAS® version 9 there is a feature - **%COPY**. Just one statement, but very powerful for said needs.

%COPY - a new statement

The %COPY statement copies specified items from a SAS® macro library. The statement accesses stored macro source code. This will write the source code to the SAS® log, or, even better, to an external file, if desired.

The syntax is: %COPY macro name /<options> out ='external file' ;
Example: **%COPY get_table / source;**

Having the ability to copy the code then permits the programmer to modify and recompile the macro as necessary. This is a new statement available in SAS® Version 9 that has made the compiled library approach much easier to use.

EXAMPLES

EXAMPLE OF A PRODUCTION MACRO CATALOG AND PROGRAMS

The following resulted from the PROC CATALOG above:

```
                    Contents of Catalog MACROLIB.SASMACR

# Name     Type          Create Date       Modified Date Description
-------------------------------------------------------------------------------
1 ACCRUAL  MACRO  30JAN2006:19:47:42  30JAN2006:19:47:42 Accrual by Months
2 EXMASTER MACRO  06FEB2006:17:28:30  06FEB2006:17:28:30 To get master
3 EXTRACT  MACRO  07FEB2006:20:52:10  07FEB2006:20:52:10 Get a table with labels
4 GET_ODB_ MACRO  30NOV2005:20:35:33  30NOV2005:20:35:33 Get all study tables
  DATA                                                 From Oracle DB
5 GET_     MACRO  19APR2005:13:18:10  19APR2005:13:18:10 Get a DB table
  TABLE
...
...
```

AN EXAMPLE OF A PRODUCTION MACRO: GET_ODB_DATA

The following is an example of a macro which downloads all of the data tables for one particular study from the main database via an odbc connection. This macro calls other macros which have already been compiled and stored in our SASMACR catalog. Along with the *get_table* macro, the *exmaster* and the *extract* macro are also used. These

two additional macros extract data from structurally different data tables in the CALGB database in a slightly different way from the *get_table* macro. Note that an external specification file is required which details the exact tables used by the study. The specification file is set up when the study is first defined and activated but can be modified at any time to customize a data download. If customization is done though, it is recommended that the original specification file be preserved and the customized file be stored under a modified file name.

This code will compile and store the macro:

```
libname mydb odbc prompt ;
options mstored sasmstore = MACROLIB ;

libname MACROLIB "N:\SAS\programs\macro_library\";

%macro get_odb_data(study=,specfile=,dest=)/store source des= "Get all
                                                study tables From Oracle DB";
libname db "&dest" ;

%include "N:\SAS\Programs\SpecFiles\&specfile."; *<== Spec file;

*Extract master tables;

%exmaster(study=&s, out=db.master);
%if &comp1 ne 0 %then %exmaster(study=&comp1, out=db.mastcomp1);
%if &comp2 ne 0 %then %exmaster(study=&comp2, out=db.mastcomp2);

*Extract samples table;

%get_table(table=sample,out=db.sample,variable_name=treat_study,variable_value=&s);

*Extract any general tables specified;

%if &death = 1 %then %extract(table=death,study=&s,out=db.death);
%if &offtrt = 1 %then %extract(table=offtrt,study=&s,out=db.offtrt);
%if &comment = 1 %then %extract(table=coment,study=&s,out=db.comment);

*Extract commmon study tables ;

%if &onstudy ne 0 %then %extract (table=&onstudy,study=&s,out=db.onstudy);
%if &dose ne 0 %then %extract (table=&dose,study=&s,out=db.dose);
%if &treatment ne 0 %then %extract(table=&treatment,study=&s,
                                                        out=db.treatment);
*Extract more specific forms;

%if &tab1 ne 0 %then %extract(table=&tab1, study=&s, out=db.&tab1name);
%if &tab2 ne 0 %then %extract(table=&tab2, study=&s, out=db.&tab2name);
%mend;

libname mydb clear;
```

To run the macro after compilation:

```
libname db "H:\CALGB\GI\80303\test";
libname mydb odbc prompt ;

%get_odb_data(study=80303,specfile=80303exspecs.txt,
                                                dest=H:\calgb\gi\80303\test);

libname mydb clear;
```

The file 80303exspecs.txt is below. This file is SAS® code defining global macro variables. Using the %INCLUDE statement, this code will be directed inserted into the program above so that all required macro variables will be defined.

File 80303exspecs.txt:

```
* List companions below for companion master file downloads;

%let comp1=150405;
%let comp2=0;

* Below are standard tables. Put 0 if download is not wanted
or table is not used on the study;

%let death   = 0;
%let offtrt  = 0;
%let comment = 1;

* Below are study specific standard tables. Provide table name
for those tables to be downloaded;

%let onstudy=otform;
%let dose=0;
%let treatment = trisfr;

* Below are specifications for study specific tables. Provide table
name and desired SAS dataset name for each table;

%let tab1 = blumma;   %let tab1name  = submission;
%let tab2 = 0 ;       %let tab2name  = ;
```

TIPS AND TRICKS

1. Create layered macro programs whenever possible. Macros should be thought of as tasks with each small task being a macro by itself, for example, the *get_table* macro. A larger task, such as getting all of the data tables needed for a study, is a series of small tasks getting one table at a time. The advantage to the layered rather than an imbedded approach is that if any changes are made to one of the small task macros, there is no need to change larger macro programming code.

2. Limit the number of variables passed into macros by creating a specification file. Too many parameters puts too much burden on the macro users. For more static situations, setting up a default specification file that is brought in using the %include statement saves a countless amount of time for the individual macro user who can use a macro in "default" mode and then customize when they are more familiar with various macro options. A specification file is also very good to have for system documentation.

3. Rename variables to common names when possible. In the example above, regardless of what the baseline, or on-study, form table name is, the SAS® dataset will be called ONSTUDY. This allows for programs down the line to be written without having to look up dataset names and to perform common tasks without having to pass the dataset name as an additional macro variable.

4. Use keyword oriented parameters so that macro call position is not vital. A call to the *get_odb_data* macro could have been written as

```
%get_odb_data(specfile=80303exspecs.txt, study=80303,
                                          dest=H:\calgb\gi\80303\test);
```

Had the macro been originally written *%get_odb_data* (study, specfile, dest) with positionally defined parameters rather than keyword defined parameters, the user must remember not just the required parameters but the exact order of the parameters to pass to the macro.

CONCLUSION

The compiled stored macro facility provides a convenient and simple means to manage multi-user macros in a production environment where integrity must be guaranteed. The use of layered macros and specification files provides flexibility, helps to insure integrity, and results in more general and widely usable production programs.

REFERENCES

Carpenter, Arthur L. (2002), "Building and Using Macro Libraries," *Proceedings of the Twenty-seventh Annual SAS Users Group International Conference*, Paper 17-27.

Carpenter, Arthur L. (2004), *Carpenter's Complete Guide to the SAS Macro Language, Second Edition,* Cary, NC: SAS Institute Inc.

Burlew, Michele M. (1998), *SAS Macro Programming Made Easy,* Cary, NC: SAS Institute Inc.

Carpenter, Arthur and Smith, Richard (2002), "Library and File management: Building a Dynamic Application," *Proceedings of the Twenty-seventh Annual SAS Users Group International Conference*, Paper 21-27.

Fu, Snow and Wu, James (2004), "Organizing and Building a Centralized SAS Macro Library," *Proceedings of the Seventh Annual Pharmaceutical Industry SAS Users Group Conference*, Paper AD11.

Sadof, Michael G. (1999), "Macros from Beginning to Mend: A Simple and Practical approach to the SAS Macro Facility," *Proceedings of the Twelfth Annual Northeast SAS Users Group*, 265-273.

SAS Institute Inc. (2004), *SAS® 9.1 Macro Language: Reference.* Cary, NC: SAS Institute Inc.

Walega Michael A.,(1996) "Creating, Maintaining, and Accessing SAS Macro Libraries," *Proceedings of the Ninth Annual Northeast SAS Users Group Conference*, 78-85.

First, Steven (2001), "Advanced Macro Topics," *Proceedings of the Twenty-sixth Annual SAS Users Group International Conference*, Paper 19-26.

Fehd, Ronald, (2005), "A SASautos Companion: Reusing Macros," *Proceedings of the Thirtieth Annual SAS Users Group International Conference*, Paper 19-26.

Stewart, Larry and Fecht, Marje (2003), "Managing SAS(r) Libraries to Improve Your Programming Environment," Proceedings of the Twenty-eigth Annual SAS Users Group International Conference, Paper 191-28.

Zdeb, Mike S. (1999) "An Introduction to macro variables and macro programs" *Proceedings of the Twelveth annual North East SAS User Group, Paper cc107.*

Timothy, Brown D. (2002) "Macro techniques for repetitive SAS Processing" *Proceedings of the Fifteenth annual North East SAS User Group, Paper ad013.*

Sanbonmatsu, Lisa (2000) "Moving from macro variables to Macro" *Proceedings of the 13 annual North East SAS User Group, Paper* bt3012.

Carpenter, Arthur L. (2005) "Storing and Using a List of Values in a Macro Variable" *Proceedings of the 30 Annual SAS Users Group International Conference*, paper cc02.

Carpenter, Arthur L. (2005) "Five Ways to Create macro Variables: A Short Introduction to the macro language" *Proceedings of the 8th Annual Pharmaceutical Industry SAS Users Group Conference*, cc02.

RECOMMENDED READING

Carpenter Arthur L. , Carpenter's Complete Guide to the SAS Macro *Language,* Second Edition, April 2004, chapter 12

CONTACT INFORMATION

Your comments are greatly appreciated and encouraged. Contact the author at:

Mirjana Stojanovic
Duke University Medical Center
(919)668-9337(phone)
e-mail: mirjana.stojanovic@duke.edu

Donna Hollis
Duke University Medical Center
(919) 681-5027(phone)
e-mail: donna.hollis@duke.edu

Dorothy Watson
Duke University Medical Center
e-mail dorothy.watson@duke.edu