Paper AD07_05

Ways to Store Macro Source Codes and How to Retrieve Them

Mirjana Stojanovic
Donna Hollis
Duke University Cancer Center Biostatistics

ABSTRACT:

This paper will focus on different ways to store macro programs. Programs developed in a multi-user clinical data management environment often experience refinement and re-programming in their lifetime, so a shared storage and usage location is valuable. This paper will present several different methods available to choose from to save macro programs.

The advantages and disadvantages of each method will be explained in detail so that users will be able to chose which method best fits their needs. %INCLUDE, AUTOCALL LIBRARY or STORED COMPILED MACRO FACILITY will be explained step by step, and examples will simplify the process of how to use these methods.

Detailed explanation and examination should provide a solid understanding of the current SAS® 9.1 environment and what this version can offer for macro usage. This presentation will explore a program developer's choices through examples and will be particularly oriented to the Windows network environment.

KEYWORDS
%INCLUDE, autocall library, compiled stored macros, SASMACR catalog, SASAUTOS, SASMSTORE, autocall libraries

INTRODUCTION

All SAS® users should know about the benefits of the SAS® Macro Facility. There is a need in almost every organization to share existing macro programs between programmers, statisticians and others. It is a very difficult to manage a lot of macro programs, stored in different places, with different functionality, yet often times very similar. If one user discovers a need for a change, it is possible that other users are dealing with the same need and, if each user changes their individual code, the result could be inconsistencies and definitely inefficiency. Changes in commonly used programs, or even just common sections of code, need to be documented and shared. To create a macro library where all macro programs will be stored in one place, is a great idea. It is time and resource saving to make one general and flexible macro program which will satisfy the needs of all relevant users in the organization. There are several ways to create and use a SAS® macro library.

%INCLUDE

This is the simplest way of calling source macro code from a macro library. It is the first step in trying to create a macro library. %INCLUDE does not set up a macro library in full, rather a %INCLUDE library holds just macro definitions. This method does not allow the saving of compiled macro code in the SASMACR catalog. In the macro library, there will be only one definition of each macro to maintain that can be used by as many programs as needed. Reusable code is stored in a file that then can be incorporated into a programmer's SAS® program with a %INCLUDE statement.

The %INCLUDE statement simply directs SAS® to where the macro code is stored. In the SAS® program, a location is specified and then the macro is included in the program as a text file. %INCLUDE is global statement and therefore not executable; global statements take effect as soon as SAS® compiles program statements. Global statements generally provide information to SAS®, request information or data, move between different modes of execution, or set values for system options.

What happens when you use the %INCLUDE statement? When you execute a program which contains the %INCLUDE statement, SAS® copies the macro code, submits and then executes the code, including any data lines that you have in the program defined by the %INCLUDE statement. The macro is stored in an external file which is then included in your current program. For example, **%include "full_path\sortds.txt ";** inserts any code in the file called **sortds.txt** into your program at the location of the %include statement**.**

Using this method, the macro must be recompiled every time a %INCLUDE is executed. This is obviously not the best way, but on the other hand it could be the first step to develop central Macro Library. All working macros can now be stored in one folder. This way, with logical naming conventions, it will be easier to find files with the certain codes.

$INCLUDE Statement Syntax:

%INCLUDE source(s) / source2=length / operating-environment-options;

Source arguments describe the location of the code you want to use with the %INCLUDE statement. One possible source is a file specification.. Source is a system option which is very helpful when using %Include.
Source2 argument causes the SAS® log to print the source statements that were included in the SAS® program. In other words, this option has the same effect as the SAS® system option SOURCE2, except that it applies only to the records that you are currently including. Specifying SOURCE2 in the %INCLUDE statement works even if the NOSOURCE2 system option is in effect.
(S2=length specifies the length of the record on the input statement.)

SOURCE | NOSOURCE
Controls whether SAS® writes source statements to the SAS® log.

SOURCE2 | NOSOURCE2
controls whether SAS® writes secondary source statements from files included by %INCLUDE statements to the SAS® log.

The downside of using %INCLUDE is maintaining the path that indicates all of the individual files. Using %INCLUDE requires you to know exactly where the physical files are stored. Sometimes, exact locations become difficult to remember. Also, a file which contains a macro code definition should contain only one macro definition and the file must contain the %MACRO and %MEND statements. SAS® statements which begin with a %INCLUDE are not a part of macro facilities. In this way, the %INCLUDE is the forerunner to a real compiled and stored macro library. Inefficiency happens in that macros called through a %INCLUDE statement will be compiled each time the %INCLUDE is executed.

Example:
/* include previously defined macro from an external file */
%include "**full_path\sortds.txt ";**

```
1          options source2 mprint mlogic symbolgen ;
2
3          libname  db       'C:\SESUGTEST\SAS_DATA_SETS\' ;
NOTE: Libref DB was successfully assigned as follows:
      Engine:        V9
      Physical Name: C:\SESUGTEST\SAS_DATA_SETS
4
5          %INCLUDE 'C:\SESUGTEST\TEXT_MACROS\sortds.txt' ;
NOTE: %INCLUDE (level 1) file C:\SESUGTEST\TEXT_MACROS\sortds.txt is file
C:\SESUGTEST\TEXT_MACROS\sortds.txt.
6         +%macro SORTDS(in=, out=, by=) ;
7         +*% Macro for sorting data set &in. ;
8         + proc sort data = &in.
9         +            out = &out. ;
10        +    by &by. ;
11        + run ;
12        +%mend SORTDS ;
```

Statement %INCLUDE incorporates the macro definition source code (sortds.txt) into the program and after the text file was included, the SAS® compiler compiled the macro definition**.**

```
57
58         %SORTDS(in=db.VISITS, out=VISITS, by=CaseID VisitNo)
MLOGIC(SORTDS):  Beginning execution.
MLOGIC(SORTDS):  Parameter IN has value db.VISITS
MLOGIC(SORTDS):  Parameter OUT has value VISITS
MLOGIC(SORTDS):  Parameter BY has value CaseID VisitNo
MPRINT(SORTDS):   *% Macro for sorting data set &in. ;
SYMBOLGEN:  Macro variable IN resolves to db.VISITS
SYMBOLGEN:  Macro variable OUT resolves to VISITS
MPRINT(SORTDS):   proc sort data = db.VISITS out = VISITS ;
SYMBOLGEN:  Macro variable BY resolves to CaseID VisitNo
MPRINT(SORTDS):   by CaseID VisitNo ;
MPRINT(SORTDS):   run ;

NOTE: There were 8 observations read from the data set
DB.VISITS.
NOTE: The data set WORK.VISITS has 8 observations and 5
variables.
NOTE: PROCEDURE SORT used (Total process time):
      real time           0.04 seconds
      cpu time            0.04 seconds
```

Calling the macro using **%sortds** will start the execution of the macro. If the macro was called before execution of the %INCLUDE statement, the program will end up with note like this:

ERROR 180-322: Statement is not valid or it is used out of proper order

Advantage: This approach was presented in SAS® at least 15+ years ago; it is an easy to use and straight forward approach.
Disadvantage: The macro definition is compiled every time the %INCLUDE is executed.

AUTOCALL FACILITY

This is another way to store macro programs. The **autocall facility** is a feature of SAS® that allows you to store the source statements that define a macro and to invoke the macro as needed, without having to include the definition source code in the program.

What is an autocall library? It is a directory containing individual files. Each file contains one macro definition. An autocall library can also be a SAS® catalog (in later versions). The name of the file must be the same as the macro name. In other words, macro names and file names must be a one-to-one match. It is also necessary that the MAUTOSOURCE option be ON and the SASAUTOS option is used along with at least one **filename fileref** statement which specifies an autocall library location. With these options in effect, when SAS® first detects a macro reference in the program, the macro facility searches the autocall library and, when a file of the same name is found, it retrieves the file from the autocall library, includes, compiles, and executes the code.

Disadvantage:

- The macro is compiled only the first time it is used in a SAS® session (any further calls will work with the compiled code). If any changes are made, it must be stored again before the changes take effect and a new SAS® session begins. Thus, if you are working on macro modifications, this is a hindrance. The complied original macro code still exists in the catalog WORK.SASMACR until the end of the SAS® session.
- Any macro language open code outside of the macro definition in the saved macro file will not be reexecuted with additional macro calls made within a SAS® session.

System options:

MAUTOSOURCE = tells to SAS® that the autocall facility is to be activated. Allows the SAS® System to search for a macro in the autocall library.

SASAUTOS = library-specification tells to SAS® where to find stored macros. The library-specification can be a SAS® fileref or the pathname in quotes. Multiple libraries can be referenced using (, , ,). [Note: if you wish to be able to reference any SAS® supplied macros, be sure to include "sasautos" as one of the libraries to be searched, i.e. SASAUTOS = (myauto, sasautos), and make sure that your user supplied macros have different names than the SAS® supplied ones.]

MAUTOLOCDISPLAY = displays the source location of the autocall macros in the log when the autocall macro is invoked.

It is recommended that storage of all autocall macros be in a location different from folders that contain other SAS® programs since they have the same extensions and it will be very hard to distinguish between them. When the above two system options are set, macros are called the same way as any other macros defined in your program but no source code needs to be included. Storing and using macros this way increases productivity by making sharing and reusing code much easier and centralizing any necessary code modifications.

To create an autocall library using a SAS® catalog the following steps are needed:

FILENAME statement(s) = to assign a fileref to the catalog that contains the autocall macros. [The name of the SOURCE name must be the same as the macro name in the library.]

OPTIONS SASAUTOS = (fileref, sasautos [optional]) MAUTOSOURCE MAUTOLOCDISPLAY [optional]

Example:

```
filename autoM “C:\SESUGTEST\AUTOCALL_MACROS\”;
options  mautolocdisplay mautosource sasautos = (autoM)  ;
```

The macro is saved in the folder with fileref autoM as **sortDS.sas** but it is not a SAS® program, it is a macro. In our program, we call the macro using **%sortDS**. Once the macro is called, we can see the source of macro code in the log .

```
3          filename autoM "C:\SESUGTEST\AUTOCALL_MACROS";
4          options mautolocdisplay mautosource sasautos = (autoM) ;
5
6          libname  db 'C:\SESUGTEST\SAS_DATA_SETS\' ;
7
8          %SORTDS(in=db.DEMOG, out=DEMOG, by=CaseID Gender)
MLOGIC(SORTDS):  Beginning execution.
MLOGIC(SORTDS):  This macro was compiled from the autocall file
                 C:\SESUGTEST\AUTOCALL_MACROS\sortds.sas
MLOGIC(SORTDS):  Parameter IN has value db.DEMOG
MLOGIC(SORTDS):  Parameter OUT has value DEMOG
MLOGIC(SORTDS):  Parameter BY has value CaseID Gender
MPRINT(SORTDS):   *% Macro for sorting data set &in. ;
SYMBOLGEN:  Macro variable IN resolves to db.DEMOG
SYMBOLGEN:  Macro variable OUT resolves to DEMOG
MPRINT(SORTDS):   proc sort data = db.DEMOG out = DEMOG ;
SYMBOLGEN:  Macro variable BY resolves to CaseID Gender
MPRINT(SORTDS):   by CaseID Gender ;
MPRINT(SORTDS):   run ;

NOTE: There were 10 observations read from the data set DB.DEMOG.
NOTE: The data set WORK.DEMOG has 10 observations and 6 variables.
```

Autocall libraries can be defined in the config.sas or autoexec.sas startup files but these are not the best place for storing SAS® macros. Macros should be kept in a SAS® autocall libraries.

The advantage of using the autocall facility is that all user-defined macros are stored in a standard location and they are not compiled until they are actually needed. The macro is stored uncompiled in an autocall library. It removes the macro definition from the calling program itself. Macros defined in separate programs must be recompiled every time that program is execute but the macro is compiled only once and then the compiled version can be reused during the SAS® session without recompilation.

STORED COMPILED MACRO FACILITY

The most exciting method of saving macros is using the store compiled macro facility. The stored compiled macro facility compiles and saves the macro source code in a permanent catalog. This compilation occurs one time, and can store the source code permanently in a SASMACR catalog. Programs can always be retrieved, the macro will work, but SAS® (macro processor) will not compile the macro again. This is the great feature of the stored compiled macro facility.

Normally, when a permanent SAS® macro catalog is not defined, the catalog is stored in the WORK library and by default compiled macros are stored in WORK.SASMACR. Creating a permanent SASMACR catalog will save time in daily production. Resources can be saved and there will be fewer errors because of duplicate programs etc. More focus can be given to saving, storing, using and reusing macros permanently stored in macro library. The compile macro is in one location, the source code could be anywhere else. It is recommended for easier maintenance to store source code in one central location, possibly in the same directory where is the SASMACR catalog stored unless complete code control is required.

The examples below will show step by step how to create a permanent macro catalog, save macros in the catalog, and how macros are called from the catalog. Also, of great importance, will be how to modify code later and how to see what was stored in the macro library. The name of library where the macro will be stored is a SAS® catalog name SASMACR. This compilation will happened only once. If the macro is called later, the macro processor executes the compiled macro.

The creation of this library need additional work before everything is setup, but later it can save a lot of programming effort and also disk space. It is highly recommended that macros be thoroughly tested before compiling to prevent any later corrections. Usually, one person should be responsible for the maintenance of the Macro Library. This doesn't mean that same person makes all of the changes in all of the stored macros, but it is recommended one person should supervise all changes in macro library ("gatekeeper"). Each company has its own policy on who will be responsible for updating and changing existing macros. 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:

- No repeated compiling of macro programs that are repeatedly used
- Possibility of displaying the entries in a catalog containing compiled macros saving macro compile 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. This time the compiled macro code will not be deleted when the SAS® sessions ends, and it can be reuse at a later time. If any changes are made to the macro source code, it must be also updated in the SASMACR catalog as well.

<u>Steps to create permanent SASMACR Catalog.</u>

1. Activate the SYSTEM options MSTORED SASMSTORE = mjstore (fileref)

- MSTORED=Allows the SAS® System to search for compiled macros stored in the SASMACR catalog. It is a system options 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.

2. Create the libname with the fileref where you want to store the macro.

- Libname MJSTORE “C:\SESUGTEST\Compiled_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).

Example
%macro sortDS (in=, out=, by=) / store source des = "get SortDS macro";

There are a few more options this time in a 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.
- **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.

Example code – Storing the macro:

```
options mstored sasmstore=mjstore;
libname mjstore “C:\SESUGTEST\Compiled_macro_library\”;

%macro sortDS (in=, out =, by=) / store source des="get sortDS macro“ ;
* Macro for sorting data set &in. ;
 proc sort data = &in.
          out = &out. ;
   by &by. ;
 run ;
%mend sortDS;
```

Using stored compiled macros

```
options mstored sasmstore=mjstore;
libname mjstore “C:\SESUGTEST\Compiled_macro_library\”;


libname db ”C:\SESUGTEST\SAS_DATA_SETS\”;        /* this is SAS® data library */
%sortDS(in=db.demog, out =demog, by=caseID Gender)
```

How do you find which macros are stored in a catalog?

SAS® answers that question with PROC CATALOG. It will list all macro programs stored in the catalog SASMACR.

```
libname mjstore "C:\SESUGTEST\Compiled_Macro_Library";

PROC CATALOG catalog=mjstore.sasmacr;
   Contents;
Run;
```

Content of Catalog :

```
                          The SAS System     15:43 Saturday, June 18, 2005   1

                           Contents of Catalog MJSTORE.SASMACR

    # Name      Type         Create Date      Modified Date Description

    1 ACCRUAL   MACRO 18JUN2005:15:43:09 18JUN2005:15:43:09 Accrual by Month macro
    2 EXTRACT   MACRO 18JUN2005:15:43:09 18JUN2005:15:43:09 Extract macro
    3 GET_TABLE MACRO 18JUN2005:15:43:09 18JUN2005:15:43:09 Get table macro
    4 LABEL     MACRO 18JUN2005:15:43:09 18JUN2005:15:43:09 Label macro
    5 SORTDS    MACRO 18JUN2005:15:43:09 18JUN2005:15:43:09 SortDS macro
```

This SAS® output will give a list of all macros stored (time, date, and description). Using the PROC SQL to get information about all compiled macros.

```
PROC SQL;
  SELECT * FROM
  DICTIONARY.CATALOGS WHERE MEMNAME IN ('SASMACR');
Quit;
```

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.

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 at any time. With SAS® version 9 there is a feature - **%COPY**. Just one statement, but very powerful for the 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 sortDS / source;

```
3          libname mjstore "C:\SESUGTEST\Compiled_Macro_Library";
NOTE: Libref MJSTORE was successfully assigned as follows:
      Engine:        V9
      Physical Name: C:\SESUGTEST\Compiled_Macro_Library
4          options mstored sasmstore=mjstore;
5
6          %COPY sortDS / source  ;
%macro sortDS (in=, out =, by=) / store source des="SortDS macro" ;
*% Macro for sorting data set &in. ;
 proc sort data = &in.
            out = &out. ;
    by &by. ;
 run ;
%mend sortDS;
```

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

There is a way to hide code when executed so that it does not appear in the log. To avoid displaying code in the log, store the code as a stored compiled macro. Because the macro is stored compiled, it cannot be seen in an editor. More importantly, the options that write information about the code to the log can be turned off in the macro. The following is a simple example:

```
libname libref 'macro-storage-library-name';
options mstored sasmstore=libref;
%macro sortDS / store;
  options nonotes nomlogic nomprint nosymbolgen nosource nosource2;
      ...more SAS® statements...
%mend;
```

By storing the code as a compiled macro, virtually no information is written to the log about the code. Only warnings and errors will be written to the log (*FAQ # 3093 SAS® Documentation*).

Before SAS® version 9 it was a problem when using compiled stored macros. Only the compiled macro form was stored by SAS®, and it was possible that you can lose the source code – or you can't remember what the macro was actually doing if you don't keep documentation on macro. Under SAS® V9, you can specify that the source code is saved with the compiled macro.

FAQ # 1616

Q: Can stored compiled macros be accessed from multiple libraries in one SAS session?

A: SAS Version 7 provides the ability to implicitly concatenate SAS catalogs. Because the LIBNAME statement allows you to logically concatenate SAS data libraries, SAS catalogs that have the same name are also implicitly concatenated. This feature enables you to reference multiple SASMACR catalogs in different SAS data libraries that have been stored in the SASMSTORE= option. SAS searches for the SASMACR catalog that is located in each library that is referenced in the LIBNAME statement and specified in the SASMSTORE= option until the entry (invoked macro) is found. The first occurrence of the entry is used.

```
LIBNAME libref <engine> (library-specification-1 <...library-specification-n>)
<options>;

  %macro-name
```

If the library specification is a path, then each path must be enclosed in single quotation marks. Here is an example:

```
libname allmine ('c:\test1' 'c:\test2' 'c:\test3');
```

CONCLUSION

The SAS® Macro Facility offers several methods to store and use SAS® macros depending on a users' needs and the state of a macro's development. The newest and most efficient method, the stored compiled macro facility, offers a relatively easy solution for the management of macros and macro libraries in individual and multiuser production environments.

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

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.

Jansen, Karl and Greathouse, Matt, (2000) "The Autocall Macro Facility in the SAS for Windows Environment," *Proceedings of the Twenty-fifth Annual SAS Users Group International Conference*, Paper 75-25.

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

TRADEMARK INFORMATION