Building and Using Macro Libraries

Arthur L. Carpenter, California Occidental Consultants

ABSTRACT

While many users take advantage of the SAS® Macro Language, few take full advantage of its capabilities to build and maintain libraries of macros. Management of large numbers of macros can be problematic. Names and locations must be remembered, changes must be managed, duplication and variations of individual macros must be monitored, and efficiency issues must be taken into consideration. These problems come even more to the forefront when macros are shared among programmers - especially in a networked environment.

Macro libraries allow you to control and manage the proliferation of your macros. Libraries can consist of a formal use of the %INCLUDE statement, a group of programs defining macros that can be automatically called, or even collections of macros that were previously compiled and then stored.

Macro libraries are extremely useful, and not really all that complicated. If you are not currently using them as part of your macro programming efforts, read on!

KEYWORDS

%INCLUDE, autocall, libraries, compiled macros, SASMACR catalog, SASAUTOS, SASMSTORE

INTRODUCTION

The use of macros is essential to an automated and flexible system. This implies that the control of the macro code is very important to the maintenance of the application. All to often macro definitions become buried within the programs that use them. The result is often a proliferation of multiple versions of similar macros in multiple programs, each with its own definition of the macro. Parallel code, two programs or macros that do essentially the same thing, is an especially difficult problem in large applications that are maintained by multiple programmers. Even when there is only one programmer, there is a tendency to clone a program ("with slight modifications"). Will each version be updated each time a bug is found? Who makes sure that the update happens? Are the various versions of the macro documented? These problems are avoided by placing the definition in a macro library.

Macro libraries are used to avoid the problem of macro cloning by providing a single location for all macro definitions. Rather than cloning the macro, it is adapted (generalized) to fit each of its calling programs and then stored in one of the libraries. Once in a library, there will only ONE macro definition to maintain, and it can be used by as many programs as is needed. Obviously this requires documentation as well as diligence. Part of the solution is to place ALL macro definitions in a common library, which is accessible to all programs in the application. This way no macro definition will be 'buried' within a program.

There are three types of macro libraries; %INCLUDE, Compiled Stored Macros, AUTOCALL Macros. Each has its advantages and disadvantages, and best of all they can be used in conjunction with each other!

It is important to understand the behavior of these libraries, how they are setup, and how they interact with each other.

%INCLUDE

The least sophisticated approach to setting up a macro library is obtained through the use of %INCLUDE files which store macro definitions. This was often the only viable type of library in earlier (pre V6) versions of SAS, and not a few SAS programmers have failed to make the transition to true macro libraries.

Strictly speaking the use of the %INCLUDE does not actually set up a macro library. The %INCLUDE statement points to a file and when the statement is executed, the indicated file (be it a full program, macro definition, or a statement fragment) is inserted into the calling program at the location of the call. When using the %INCLUDE to build a macro library, the included file will usually contain one or more macro definitions.

Although you can identify the file to be included directly from within the %INCLUDE statement, it is usually done indirectly through the use of a *fileref*.

```
filename macdef1
      'c:\mymacros\macdef1.sas';
........
%include macdef1;
........
%macdef
........
```

One way to build a library or collection of files that are to be 'included', is to place them in one central location. This way they will be easier to find and maintain. Some users of include libraries will indicate that these file are to be 'included', and therefore might not be complete programs, by using an extension of INC instead of SAS. While using an extension of INC in no way hampers the file's use by SAS, windows users should be aware that the file will not be marked by a SAS registered icon.

As was mentioned above the included file can contain any snippet of SAS code. Within the context of this paper however, we are interested in building macro libraries and to do this the included file would contain a macro definition *e.g.* `%MACRO` and `%MEND` statements. There are a couple of efficiency issues that the user of include libraries should remember.

Generally a given file should not contain more than one macro definition, and the macro being called should not be called from within the included code. The first is important because if the file contains multiple macro definitions, then all the definitions must be loaded, and compiled, just to use one of the macros. Of course if all the macros will eventually be used this does not really matter. Secondly if the macro call is placed within the included code, the code will need to be re-included (and the macro recompiled) each time the macro is to be used.

The greatest disadvantage of using the %INCLUDE in a large application is tracking and maintaining the *filerefs* that point to the individual files. While this is less of a problem in static systems, it is still non-trivial. This issue virtually goes away with the use of the other macro library alternatives discussed below.

COMPILED STORED MACROS

Macros are always compiled before they are executed and it is possible to store the compiled code for future use. Compiled macros are stored in a catalog named SASMACR, and by default this catalog is stored in the WORK library. Each compiled macro is stored with the entry type of MACRO and with an entry name corresponding to the name of the macro. When a macro is called, SAS automatically searches this catalog for the compiled version of the macro that was called. Permanently compiled stored macros are also written to a SASMACR catalog, but this catalog is stored in a different (permanent) library.

The ability to store and make use of compiled stored macros is by default turned off. The user turns on the abilty to make use of compiled stored macros with the system option

MSTORED (NOMSTORED turns it off). The library that is to be used to store the permanent SASMACR catalog is specified by using the SASMSTORE= option.

The following OPTIONS statement turns on the use of compiled stored macros and designates the two *libref*s PROJSTOR and ALLMSTOR as the locations that are to be searched for the compiled stored macro catalog. If the compiled macro is in more than one catalog, the definition found first will be used (read left to right).

```
options mstored sasmstore=(projstor
allmstor);
```

By default the compiled macro is stored in WORK.SASMACR. You redirect this location at compile time through the use of the /STORE option on the %MACRO statement. For the SASMSTORE definition above, the macro AERPT shown below, will be stored in the PROJSTOR.SASMACR catalog.

```
%macro aerpt(dsn, stdate, aelist) / store;
```

The use of compiled stored macros is not without problems. The developer must make sure that a macro does not exist in more than one catalog or if it does (on purpose, of course) that the search order of the catalogs is correct. Also, since it is not possible to reconstruct the code used to create the compiled macro, the developer must make sure that the code is correctly maintained independently from the SASMACR catalog.

This highlights a second problem for these libraries. The compiled macros are all in one location, the SASMACR catalog, however the code itself could be anywhere. Usually developers that use compiled stored macro libraries will also have a central location to store the source code. A logical location is in the same directory as the SASMACR catalog.

AUTOCALL FACILITY

When a requested macro is not found in a SASMACR catalog, the AUTOCALL library is searched. Much like an %INCLUDE file, this library contains the macro definitions in the form of SAS code. When a macro is called, SAS searches for a file with the SAME name as the name of the macro in the specified AUTOCALL location. The code in the corresponding file is then submitted for processing. Since this file contains the macro definition, the macro is then compiled and made available for execution. Under Windows this location is a folder and each macro definition is an individual file. Under MVS a PDS is used with each member corresponding to a macro.

By default the ability to make use of the AUTOCALL facility is turned on. The following code makes sure that the autocall facility is available (MAUTOSOURCE) and specifies the ***FILEREF***s of the locations (SASAUTOS=) that contain the SAS programs with the macro definitions.

```
options mautosource
        sasautos=(projauto allauto);
```

Although the documentation is not entirely clear on the subject (or sometimes even incorrect, see *Carpenter's Complete Guide to the SAS® Macro Language*), be sure that you use *filerefs*, NOT *librefs*, to specify the locations of the autocall programs (Burlew, 1998, correctly specifies the locations with the FILENAME statement). You can also replace the *fileref* in the SASAUTOS= option with a direct reference, however this approach is less flexible and is not as 'clean'.

MACRO SEARCH ORDER

As these libraries of macros are established it is important for both the developer and the user to understand the relationship between them. One of the more important aspects of this relationship is the order of locations that SAS searches for a

macro once it is called.

When a macro is called, the macro facility must first find the macro definition before it can be inserted for execution. Since the macro definition can be located in any of several locations, the developer needs to control the search. The search for the macro definition uses the following order.

1) WORK.SASMACRO
2) Compiled Stored Macros
3) Autocall Macros

The first time that a macro from the Autocall library is called it will not be found in any of the catalogs of compiled macros, but once called it will be compiled and the compiled code stored (in WORD.SASMACR unless otherwise specified). On successive calls to that macro its compiled code definition will be found and the search will not extend to the Autocall library a second time. This process minimizes the compilation of the macro.

This is a major advantage over the use of the %INCLUDE as a macro library. When a macro definition is brought into the program through the %INCLUDE, it will be compiled for each %INCLUDE. Of course if the programmer is careful, this is not such a bad thing. If the %INCLUDE appears only once, the macro will be compiled at that time and added to the WORK.SASMACR catalog and the compiled macro definition will be used from then on (as long as it is not included another time).

MACRO LIBRARY STRUCTURE

Placing your macros in a library will help to organize you programs. However in larger groups, projects, or in organizations with multiple programmers sharing macros, it becomes necessary to organize the libraries themselves. In the designations of the locations of both the Autocall library (SASAUTOS=) and the compiled stored macros (SASMSTORE=) shown above there are two locations. By specifying multiple locations for the libraries it becomes possible to organize them into collections of macros. Typically for the work that I do, I like to arrange the collections according to how the macros are to be used. Macros that are specific to a task or project will be placed in the location that will be searched first. Then the macros that are more generalized or are useful to multiple projects are placed in a location that will be available to users of all projects. This arrangement allows the developer to create general tools that are available to everyone as well as specific macros that apply to only one project or task.

MACRO LIBRARY STRATEGY

The question is then not IF to set up a library, but which macro library setup to use.

The %INCLUDE library and the AUTOCALL library can be set up to in a very similar fashion. The primary advantage of the AUTOCALL library is that the developer does not need to manage or work with the individual *filerefs*, as these are controlled automatically through the SASAUTOS= system option.

Unless a macro is unusually large or complex, it generally takes very little time to locate and compile a macro stored in the AUTOCALL library. This suggests that there is not a substantial time savings in using the compiled stored macro library. Since both libraries require the developer to store and maintain the source code, there does not seem to be a compelling reason to adopt one library type over the other. For this reason and because everyone already uses the AUTOCALL facility anyway to get to the SAS supplied autocall macros, the AUTOCALL library seems to be used much more frequently than the compiled stored macros library.

One strategy that has worked well in limited situations, combines these two types of libraries.

This combined library approach specifies an autocall library (SASAUTOS=). It also turns on the compiled stored macro library (MSTORED) and then points the location (SASMSTORE=) to the SAME location as the autocall library. Since the SASAUTOS= *fileref* and the SASMSTORE= *libref* both point to the same directory, the SASMACR catalog will reside in the same location as the source code that defines the macro. Once this is done all the macros in the autocall directory will ALSO have the /STORE option. Now we have the best of both worlds. One source program and a compiled macro all stored in one easy to find location.

INTERACTIVE MACRO DEVELOPMENT

When developing macros in the interactive environment, special care must be taken to make sure that the correct macro definitions are compiled and stored. If the developer is not careful it is possible to call a macro without using the latest update to the macro definition. This is not a good thing.

As was discussed earlier, after a macro definition is submitted, the compiled code is stored in the appropriate SASMACR catalog. This catalog will be in the WORK library unless compiled stored macros are being used and the /STORE option is present on the %MACRO statement. When the macro is then later called, SAS looks for the macro definition in one or more of these catalogs and only if the definition is NOT found is the AUTOCALL library searched.

Suppose the developer is debugging a macro whose definition resides in the AUTOCALL library. She calls the macro for execution, but is not satisfied with the results. If she edits the program, saves it back into the AUTOCALL library, and then re-executes it, the results will be the same! Her changes will be ignored! Her changes are not implemented because the macro has not been recompiled! She has updated the code, but because the macro has already been compiled and resides in the SASMACR catalog, SAS will never look for the new definition in the AUTOCALL library. She needs to do one of two things after making the change in the macro definition. She can:
1) submit the macro definition (%MACRO to %MEND), this places the latest definition in the SASMACR catalog.
2) delete the compiled macro entry from the appropriate SASMACR catalog. The next execution of the macro will cause SAS to seek out the AUTOCALL definition, which will then be compiled and re-stored in the SASMACR catalog.

When your SAS environment includes macro libraries, you must remember that as you edit your macro definitions, you must also update the appropriate SASMACR catalogs as well. It is not enough to just change the macro definition.

The above example illustrates a situation that is usually not an issue when using a %INCLUDE library to hold macro definitions. When the %INCLUDE brings in a macro definition, the definition itself is usually inserted directly into the code, it is then submitted and compiled - all in the same operation. Although this may initially sound like an advantage, it actually is not because the macro must be recompiled each time the %INCLUDE is executed. This method does not take full advantage of the ability to save the compiled macro in the SASMACR catalog.

AUTOCALL MACROS SUPPLIED BY SAS

SAS has supplied a number of macros with the SAS System *e.g.* %LEFT, %LOWCASE, and %VERIFY. These macros are actually supplied as macro code, and the code is placed in the SAS AUTOCALL library. If you want to make use of these macros, and you do, you must include this library in your list of locations search

by the autocall facility.

By default the automatic *fileref* SASAUTOS is available. An options statement setting the AUTOCALL options to the defaults would be:

```
options mautosource sasautos=sasautos;
```

If you specify locations for AUTOCALL libraries, you need to make sure that the SAS AUTOCALL library is also specified. Failure to do this will make ALL of the SAS autocall macros unavailable. In the example of the OPTIONS statement in the AUTOCALL section of this paper, the SASAUTOS *fileref* is NOT included and the autocall macros provided with SAS will NOT be available. The OPTIONS statement should be rewritten as:

```
options mautosource
        sasautos=(projauto allauto
sasautos);
```

Notice that the SASAUTOS *fileref* is listed last. This allows the user to create macros that will override the default definitions of the macros supplied with SAS.

SUMMARY

Macro libraries allow the macro developer to store, maintain, and manage large numbers of macros. The two types of macro libraries (AUTOCALL and Compiled Stored Macros), were introduced in V6. These libraries allow the user to avoid the use of the %INCLUDE statement to bring in macro definitions, while efficiently accessing and utilizing the power of the macro.

Using macro libraries is not difficult nor is it overly complicated. Both styles of libraries are established through the use of the OPTIONS statement, although the AUTOCALL macro facility is available by default.

Macro libraries foster an environment that promotes good macro programming practices by making it easier to avoid the use of duplicate macro definitions. As a result of using these libraries you will find that it is easier to track and maintain your macros, and macro programming will become even more fun.

ABOUT THE AUTHOR

Art Carpenter's publications list includes three books *Quick Results with SAS/GRAPH® Software*, *Annotate: Simply the Basics*, and *Carpenter's Complete Guide to the SAS® Macro Language*, and numerous papers and posters presented at SUGI, PharmaSUG, NESUG, and WUSS. Art has been using SAS since 1976 and has served in various leadership positions in local, regional, and national user groups.

Art is a SAS Certified Professional™, and through California Occidental Consultants he teaches SAS courses and provides contract SAS programming support nationwide.

AUTHOR CONTACT

Arthur L. Carpenter
California Occidental Consultants
P.O. Box 6199
Oceanside, CA 92058-6199

(760) 945-0613
art@caloxy.com
www.caloxy.com

REFERENCES

Burlew, Michele M. 1998, *SAS® Macro Programming Made Easy*, Cary, NC: SAS Institute, Inc., 280pp.

Carpenter, Arthur L., 1998, *Carpenter's*

Complete Guide to the SAS® Macro Language, Cary, NC: SAS Institute, Inc., 242pp.

TRADEMARK INFORMATION