CREATING, MAINTAINING, AND ACCESSING SAS MACRO LIBRARIES

Michael A. Walega
iBRD-Rostrum Global, Inc.

You've finally done it; you completed the programming for the World's Greatest SAS Macro. It is the end all to all SAS programs written to date by all programmers. Your satisfaction is immense; you can't wait till other users feel the raw power of your program as they put it through its paces.

You present your Macro program to your supervisor. He reviews it and smiles, agreeing with you that, yes it is the World's Greatest Macro program. But wait, he says; I have a couple of questions for you.

- How do other users gain access to your program? Do they store copies of it in their own directories, or will there be a central location for the Macro program?
- Is the coding style that you have followed sufficiently simplified and documented such that others can follow it?
- Has the program been validated in a standard way?
- Is this program sufficiently generic such that all users can benefit from its availability, or is this beneficial for a specific user base?
- Is this the only version of this program that you intend to make available? What about future modifications to the code?

You cringe as you realize that you have no answers to these questions. He suggests that you discuss your macro with the SAS Macro Library Manager (MLM). You agree, and set off to seek the MLM.

While this scenario may seem a little surreal, it is not far removed from reality. Macro programs that are designed for multi-user access should be stored in a central location, a program warehouse (to employ that buzzword du jour), rather than having everyone store a copy in their individual user accounts. This common program warehouse is known as a SAS Macro Library.

To assure that any Macro program that is destined for a program warehouse is appropriate for multi-user access, and to assure consistency, the following issues should be addressed prior to implementation of a Macro Library:

- Have ground rules been set to assure that the Macro program meets all appropriate criteria?
- Has the program been developed with sufficient generic functionality in mind such that many users will benefit from its availability?
- Has a process for version control been implemented?
- Has a SAS Macro Library been set up in an area that is available on a read only basis to users?
- What about more than one program warehouse? Is it necessary?

Successful implementation of a system-wide SAS Macro Library depends on a well-designed process that addresses all of the above issues. This paper will attempt to expand on these critical areas, as well as other aspects of setting up SAS Macro Libraries. This will include management/location of Macro Libraries, validation, documentation and inclusion of programs in the Macro Library and user access to Macro Libraries. As a prelude, this paper will provide guidelines on the functionality and coding techniques that apply to all Macro programs destined to reside in a program warehouse.

Macro Functionality

It is important to recognize that any programs that are placed in a system-wide Macro Library must be sufficiently generic such that the maximum number of users benefit from their use. While Macro programs come in all shapes, sizes and colors, their functionality can usually be broken down into two groups: low-level, single function utility macros, and high-level, multi-step macros.

Low-level utility macros should be kept simple. Values returned by these macros should be limited to one value. Examples of low-level Macro programs include: calculation of age, conversion of weight in pounds to kilograms, calculation of depreciation, determination of the number of arguments stored in a macro variable, etc.

In most cases, positional parameters should be used in the calls to the Macro program.
Defaults can be supplied as part of the macro call. An example is shown below:

%age(startdt, dob, age);

In this example, startdt and dob are the SAS variable names for the dates used to calculate the value of age. These are passed to the Macro program. The variable age is the name of the SAS variable that contains the calculated result returned by the Macro program. If this were the default SAS variable name used by the Macro program, then another way to call this macro would be

%age(startdt, dob);

High-level, complex macros consists of code capable of returning multiple values, creating data sets, or creating data displays. These may consist of several DATA steps and/or PROCs. Examples of high-level macros include: generation of change from baseline summary tables for clinical laboratories, generation of sales figures by account exec/region/quarter, energy use by location, generation of collapsed data sets, generation of data libraries from ASCII files, etc.

Because of their potential complexity, calls should be made using keyword parameters. In this way, the user knows the options that have been selected, and the call to the macro resembles SAS language code. Unlike the low-level macros, no defaults should be supplied. Again, the complexity of these macros dictates that all options be user entered. Examples are shown below:

%crunch(ds=efficacy, group=tmI, patnum=pid, var=eff1);

%makelib(datpath=m:\proj\data\raw, compath=m:\proj\data\raw, libpath=m:\proj\data\sas, ext=dat);

**Macro Coding Guidelines**

While everyone has their own style of programming, it is important to remember that Macro programs stored in a system-wide Macro Library should be relatively consistent in their style and content. The reason for this is twofold. First, because these programs are available to many users, a common coding style benefits those who are interested in reviewing the code to gain an understanding of how the macro program works. Secondly, a common, simplified coding style benefits those who have to modify/update the code, especially if they are not the original authors. A simplified coding style includes the following common elements:

- Thorough documentation throughout program
- Lots of white space between lines/blocks of code
- Indentation
- Lowercase SAS Language code in high-level Macro programs
- UPPERCASE macro variables (&MACVAR) in high-level Macro programs
- No Hardcoding!! (Data-driven code only)

In addition to a simplified coding style, there are several other coding issues that need to be addressed. These include modification of the structure of the program invoking the macro, deletion of interim data sets, and options to provide intermediate data dumps in high-level macros.

When developing system-wide Macro programs, one must always keep in mind that the Macro program must not modify the structure of any part of the program invoking the macro. This includes variables, data sets and macro variables located within the program that invokes the macro. A good rule of thumb for any system-wide Macro program is that all macro variables within the Macro program are local, not global. Any macros that are declared as global within a Macro program have the potential for modifying the value(s) of macro variables located outside of the program. This is especially true for interactive SAS users, such as those that run SAS in a PC environment. Here the tendency is to run multiple programs from a single SAS session, such that the chances of overwriting macro variables already defined increase dramatically.

If a system-wide Macro program employs DATA steps or PROCs, the macro must not modify the data set(s) of the program that invokes the macro. If a Macro program is developed that results in modification of data in the calling program, then the data sets used in the Macro program must have unique names. One way to circumvent this problem is to have all data set names in the system-wide Macro program begin with z underbar (Z_). This naming convention is also recommended for SAS language variables that are created in the system-wide Macro program; that way, they will not be in conflict with variable names within the calling program.
Interim data sets created by a system-wide Macro program should be deleted prior to the end of the Macro program; PROC DATASETS can be used to accomplish this task. Again, any potential conflict with the calling program's structure is minimized through the use of this basic PROC.

```
proc datasets library=work memtype=data;
  delete z_dsn;
  quit;
run;
```

Lastly, it is suggested that a switch in the Macro program be available to provide users of high-level Macro programs with the ability to produce intermediate data dumps. This allows the user to review output to assure that 1) the macro is working the way they want it to, and 2) their data are in the correct format for the macro program to effectively complete its task.

### Management of Macro Libraries

To best manage a system-wide SAS Macro Library, it is suggested that one individual be designated as the Macro Library Manager (MLM). This individual acts as the gatekeeper to the Macro Libraries. Their responsibilities would include controlling access to the Macro Libraries, macro version control, adherence to standard guidelines/procedures for programs submitted for inclusion in the Macro Library, and overall maintenance of the Library. It would be the responsibility of this individual to approve all programs submitted for inclusion, to publicize the availability of new programs in the Macro Library, and to work with authors on modifications to programs already in the Macro Library.

### Location of Macro Libraries

The location of SAS Macro Libraries on a hardware platform is driven by several parameters:

- The actual hardware platform(s) involved,
- User availability, and
- System functionality.

Ideally, the location of Macro Libraries should be transparent to any/all users that require access to the library. Single platform environments are straightforward: provision of an appropriate path name to the user (plus any additional necessary commands to access the libraries) should be all that is required.

The challenge of setting up SAS Macro libraries occurs when multiple hardware platforms are used to run SAS. An example is the mainframe server/PC client environment. In this type of environment, users may develop their programs on a PC/LAN desktop, while production runs of programs is accomplished on a mainframe (e.g., VAX/VMS, IBM/MVS, SUN/UNIX). In this instance, location of the Macro Library is critical; users from all platforms must have easy access to the programs stored in the library.

One way to accomplish this task is to set up a driver program that initializes when SAS is launched and creates the appropriate path name to the Macro Libraries. The system macro variable SYSCCP contains information on the operating system platform that SAS is currently running on. For example, if a user launches SAS on a PC, the value of SYSCCP is 'WIN'. If a user launches SAS on a VAX platform (not ALPHA), then the value of SYSCCP is 'VMS'. This information can be used to set correct path names to point to the locations of Macro Libraries, as shown below.

```bash
%if %upcase(&SYSCCP) = WIN %then %do;
  %let mainpath = m:asasmacros;
  %let projpath = m:mydirmacros;
%end;
%else %if %upcase(&SYSCCP) = VMS %then %do;
  %let mainpath = disk1:asasmacros;
  %let projpath = disk1:mydirmacros;
%end;
```

The driver program can also be used to limit access to Macro Libraries by certain users. The system macro variable SYSJOBID contains the name of the currently executing batch job or the userid that invoked the current session. Note that the value this macro variable contains is dependent upon the operating system. An example of its use on the PC platform is shown on the next page.
%if %upcase(&SYSJOBID) = IS85 %then
  %let proopath = m:\is85\macros;
%else if %upcase(&SYSJOBID) = IS82 %then
  %let proopath = m:\is82\macros;
options sasautos="(&MAINPATH", "&PROJPATH");

Validation of Macro Programs

Prior to inclusion into a system-wide Macro Library, the macro programs must be validated. Although the validation process can take many forms, the following has been successfully implemented by the author.

First, the Macro program should pass through a peer-review process. The author of the Macro program, and an unbiased peer should review the code with respect to accuracy, completeness of error capture, and intent of the Macro program. This process gives the author an opportunity to explain the rationale behind his World’s Greatest Macro program, as well as explain and/or justify any assumptions that he may have made during the development of the Macro program. For example, the author of a program that calculates age may use the formula

\[
\text{age} = \frac{\text{obsdate} - \text{DOB}}{365}
\]

while everyone else in the company calculates age as

\[
\text{age} = \frac{\text{obsdate} - \text{DOB} + 1}{365.25}
\]

Although the difference between the two algorithms is subtle, in certain instances this difference can be critical.

After the Macro program has been through the peer review process, the code must be successfully applied to at least two (2), and preferably three (3) different “projects”. It is up to the user to define “projects”. In the pharmaceutical industry, this usually means application of the Macro program to data from two or three different clinical trials (e.g., a vaccine trial, a CHF trial, an asthma trial). Comparison of the output generated by the Macro program to previously validated output from these projects is sufficient to serve as documentation of a successful validation. The validation documentation (hardcopy of source code, log files, output) then becomes an integral part of the documentation for the Macro program, as described in more detail below.

Documentation

Documentation can take one of two forms: external or internal to the program. It is ABSOLUTELY IMPERATIVE that a Macro program be documented internally. One can NEVER be accused of overkill when it comes to documenting a program. The internal documentation should include, at a minimum,

- Name of the program,
- Author’s name/location (i.e., extension)
- Location of Macro on system
- Purpose of Macro
- Date first developed
- List of all macro parameters, optional and required, plus a brief explanation (1-2 lines) of their function

In addition, external documentation must be provided. The external documentation should provide a reviewer a detailed history of the Macro program. Following is a list of suggested items for inclusion:

- Name of the program
- Version number
- Author’s name/location
- Purpose of Macro program
- Date first developed
- Clear description of any input data set(s) with all requirements and restrictions noted
- Clear description of macro parameters, listing defaults, requirements, restrictions
- Commented sample calls to Macro program
- Description of any output data set created by the Macro program
- List of all other Macro programs called by the Macro program, and their location
- Hardcopy of source code
- Sample output
- Validation documentation, to include
  - Hardcopy of logs and output from validation runs
  - Name(s) and location(s) of data sets used to validate code
Inclusion of Macros in Macro Library

Several steps need to be completed prior to inclusion of the World’s Greatest Macro to in the system-wide Macro Library. After the validation steps have been completed, the author of the Macro program presents all appropriate documentation to the individual designated as the MLM. To make streamline this process for the MLM, a checklist that documents the required steps prior to inclusion in the Macro Library is suggested (see Appendix 1). In this way, the MLM can accumulate additional documentation on the Macro programs stored in the system-wide Macro Library. In addition, this checklist can be returned to the author for further work if specific tasks were not completed. Items that might be included on the checklist include:

- Coding standards followed?
- Calling structure modified?
- Interim data sets deleted?
- Unique macro parameters?
- Intermediate data dumps (if necessary)?
- Validation steps (peer review, comparison with known results) completed?
- All documentation (internal/external) in order?

When all of the items on the checklist have been supplied by the author, the MLM can then sign off on the checklist, and the World’s Greatest Macro can then take its rightful place in the system-wide Macro Library!

A copy of the signed documentation is returned to the author(s); the original becomes part of a ‘SAS Macro Library’ documentation binder. The binder contains a copy of all SOPs, guidelines, etc. that pertain to the system-wide SAS Macro Library, as well as documentation for each Macro Program in the library.

Version Control

One of the more difficult challenges facing anyone the designated MLM is the issue of version control. As soon as a Macro program is made available to users, someone will attempt to ‘fix it even though it ain’t broke’ is wonderful and horrifying at the same time. This leads to the following Macro Library axiom: At all times, one and only one (1) copy of a macro program will exist in the system-wide Macro Library. So long as a version control process is in place, then updating of currently available Macro programs is both acceptable and welcome.

If indeed a Macro program in the library is in need of updating, the first step is to determine why. Is it an error that was not captured during validation? Is it obvious (to the user, anyway) that a simple embellishment to the program will dramatically increase its usefulness? Whatever the reason, the MLM should receive in writing documentation that indicates the reason(s) for the desired modification of the Macro program. This documentation would then be placed in the Documentation binder and serve as notice for initiation of the macro revision process.

If available, the original author of the macro would be the logical person assigned to the task of revising the code. If unavailable, the individual responsible for peer review of the Macro program during validation would be assigned to the task.

After the code has been updated, it must go through the same validation process as the original Macro program. Once this has been completed, and once the MLM has signed off on the Macro as being ready to include in the system-wide Macro Library, the following process (or similar) should occur.

The original Macro program should be moved to a subdirectory within the system-wide SAS Macro Library directory. This subdirectory (such as OLDMACS) serves as a repository for previous versions of Macro programs. The original version (or previous version) would be copied to this subdirectory. The name of the program, instead of being macroname.SAS, would be changed to macroname.OLn, (n = 0,1,2...), where macroname is the name of the Macro program and n is the version number of the macro being saved to this subdirectory. For example, macroname.OL0 would be the original version of the code, macroname.OL1 would be the first revision of the code, etc. This way, the original program, as well as any subsequent updates, are always available for review. So too with the documentation; all versions of the documentation should be saved, such that a paper trail of the Macro program from initial to current version exists.
Multiple Macro Libraries

While some purists may cringe at the thought of having more than one Macro Library, in many cases multiple Macro Libraries are extremely useful. As long as users are able to discern between the libraries (as described below), multiple libraries are not difficult to manage.

Multiple libraries can take the form of generic, system-wide libraries (which is the main focus of this paper), local, project-specific libraries, or some combination of the two. Project-specific libraries may contain project-specific algorithms, titles/footnotes, etc. The Macro programs located in this type of library are not generic enough to be included in the system-wide library, but are still useful enough in macro form to be included in its own macro library.

Macros stored in a project-level library should be managed by the users working on the project. To maintain consistency, it is suggested that any Macro program placed in a project-level library be subjected to the same guidelines for coding and internal program documentation.

If multiple system-wide Macro Libraries are employed, it is recommended that each library contain distinct types of Macro programs. For example, one library could contain all low-level, utility macros, while another could contain all high-level macros. This division of programs would minimize any chance of the same program names being used in two different places, each with two distinctly different functions! Management of multiple system-wide Macro Libraries is a challenge for the MLM. Although many of the ideas presented here also pertain to multiple system-wide libraries, the design and implementation of these libraries requires significant planning and development.

User Access to Macro Libraries

As previously discussed, version control is best implemented by granting one person (Macro Library Manager) write access to the Macro Library. All other users should be granted only read access; again, this may be modified if necessary to accommodate multiple Macro Libraries. Actual access to SAS Macro Libraries by users is a straightforward process, and is accomplished through the use of the Autocall Facility.

The Autocall Facility is a technique that enables storage of macro source code (programs) in external files, and allows users to invoke these macros without the need to either define or include them in the current program. The Autocall Facility is used to retrieve macros that are stored in SAS Macro Libraries. Access to these Libraries is accomplished through the use of SAS System options found in the OPTIONS statement (Note that these options can also be invoked in a configuration file). These options are MAUTOSOURCE, SASAUTOS and MRECALL. These options are described below in more detail.

The MAUTOSOURCE option controls the availability of the Autocall Facility. With the MAUTOSOURCE option enabled, a macro that is invoked in a program (but not yet compiled by the Macro Processor) is searched for in each of the Autocall libraries specified by the SASAUTOS option by the Macro Processor.

As with permanent SAS data sets and catalogs, the location of a Macro Library must be explicitly stated such that SAS can search the appropriate directories/path names for the requested macro(s). The SASAUTOS option is used to indicate the path(s) to be searched. The syntax is SASAUTOS=(pathname1, pathname2,...). Multiple path names can be used; SAS searches these path names in the order that they are given in the SASAUTOS option. (Ideally, the SASAUTOS option is invoked only once during each SAS session. This will minimize the possibility of looking for like-named (but structurally different!) macros in different directories.

These path names can take the form of

- a SAS fileref
- the name of an aggregate storage location on the hardware platform, enclosed in quotes
- a mixture of filerefs and aggregate storage location names.

Conclusions

The steps required to develop and maintain a system-wide SAS Macro Library are numerous. Much up front work is required to assure that
users can effectively use generic macro programs. Once processes are in place to guide users in access to Macro Libraries, development of programs for inclusion in Macro Libraries, and maintenance of Macro Libraries, the benefits to be reaped by both users and their company are enormous.

The author can be contacted at

iBRD-Rostrum Global, Inc.
Suite 100, Gwynedd Hall
1777 Sentry Parkway West
Blue Bell, PA 19422
215-540-8400x491

SAS is a registered trademark of the SAS Institute, Cary, NC.

Reference

## APPENDIX 1

### Checklist for SAS Macro Programs

**MACRO NAME**

**AUTHOR**

**DATE SUBMITTED**

**TYPE OF MACRO**

<table>
<thead>
<tr>
<th></th>
<th>Approve</th>
<th>Reject</th>
<th>Reasons for Rejection</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. CODING STANDARDS</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>1.1 ALL MACROS</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>1.1.1 CODE FORMAT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.1.2 CALLING STRUCTURE MODIFIED</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.1.3 DELETE INTERIM DATASETS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.1.4 CORRECT MACRO PARAMETER CALLS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.1.5 OTHER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.2 HIGH LEVEL MACROS ONLY</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>1.2.1 INTERMEDIATE DATA DUMPS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.2.2 OTHER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. VALIDATION</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>2.1 PEER REVIEW</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.2 COMPARISON WITH 2 OR 3 PROTOCOLS (CIRCLE ONE)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.3 OTHER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. DOCUMENTATION</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>3.1 EXTERNAL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.2 INTERNAL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.3 OTHER</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**The macro has been accepted for inclusion in the macro library.**

**The macro has been rejected for the reasons stated above.**

**Signature of Manager of Macro Library**

**Date**

---

85

**NESUG '96 Proceedings**