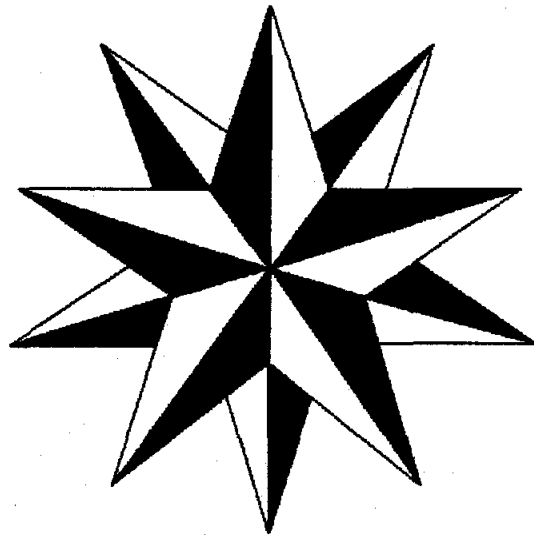


EFFECTIVELY USING THE SAS MACRO FACILITY



**Howard Levine
Levine Software Systems**

Effectively Using the SAS® Macro Facility

Howard Levine, Levine Software Systems

Introduction

SAS Macros give systems developers the ability to build easily maintained and sophisticated systems that can be built quickly and easily with the SAS® System. The key, of course, is to make extensive and thoughtful use of the macro language.

The macro language is well-suited for generating repetitive code and for conditionally executing code. In addition, the use of parameters allows even greater flexibility. Therefore, macros can generate standardized code that is made more flexible by using parameters. Flexible macros are useful as building blocks for systems. Such macros can be used in a variety of situations by changing parameters such as input and output data set names, variable names, date ranges, and various selection criteria. More simply, *well designed macros with parameters are good system building blocks.*

It is critical when building macro-based systems to build macros that are useful building blocks for writing other macros and SAS programs. There are essentially *two types of building blocks: Tools and Applications.*

A *tool* is a macro that can be used in any system because of its general usefulness. An example is an %OBS macro that determines how many observations are in a SAS data set.

An *application* is a macro that is useful only in a certain sub-system or project. An example of an application is a macro that prints out a report in a particular format. Such a macro might have to be used extensively for a particular project, but it unlikely to be of general use for all projects.

What is a Macro-based System

A macro-based system consists of application macros that may invoke other application and tool macros and generate SAS statements. All SAS statements are generated from macros. There is no "loose" SAS code which exists outside of a macro definition. The basic idea is that coding is changed from writing SAS statements to creating building blocks and then putting them together.

Why use them?

There are several advantages to using building blocks instead of having each programmer code

everything himself every time he is building a system. Some of them are listed below.

- It is easier to simply invoke a macro rather than re-write the code every time it is needed.
- Expert coding that beginning SAS programmers could not understand can be used by everyone just by invoking the macro written by the expert.
- Changes to systems can be accomplished by making fewer coding changes - change one macro rather than all your programs.
- Programs are easier to read and understand because macros become a user defined language. It is a higher level language than the SAS program which is generated from the macros. This is true in the much same way that the SAS System is a higher level language than PL/I or C.

Some Examples

Some contrived, but useful, examples will illustrate these points.

The following three line program reads data, processes it, and then writes a report.

```
%READDATA(DDNAME=DATAIN, VAR= NAME $  
DATE AMOUNT, OUT=DATA1)
```

```
%PROCESS(CLASS=DATE,  
VAR=AMOUNT,DATA=DATA1, OUT=DATA1)
```

```
%REPORT(DATA=DATA1, FILE=PRINT)
```

Just from looking at the macro names, it is obvious what each macro does in a general way. The keyword parameters provide much information about what the macro will be doing during that invocation of the macro. Once the macros are built, it is very easy to write programs that would have previously required much more coding. Also, the lack of clutter and good use of keyword parameters make the code easier to understand.

The next example shows how a macro can allow a change in one place (the macro) to affect every relevant program. Pretend that you are responsible for a reporting system that has many programs generating reports about snowfall and snow storms. You would want to be able to easily change the definition of a storm in all of your reports at once rather than change each one

individually. Not only does this mean less work for you, but it also insures consistency among reports.

```
%macro      snostorm(storm=snostorm,
                    yes=1,no=0);

if (season="SUMMER" and snowfall gt 1)
    or (snowfall gt 12) then &storm = &yes ;
else &storm = &no;

%mend snostorm;
```

The snostorm macro, if used in all reports pertaining to snow storms, permits the definition of a storm to be changed for all programs in just one place.

The following example shows how macros with parameters can be very flexible building blocks for programs. The macro SALES creates a data set containing sales for the last N PERIODs. Since the period can be any of the periods used with the INTNX function, it is possible to create different data sets with the same macro. The macro enables a programmer or end user to easily specify the kind of data set desired. Another advantage is that all the data sets created with this macro should be consistent. That is, adding January, February, and March totals for a monthly data set will give you the first quarter total. If you write separate programs for each type of period, it is possible that there would be small differences in how the data sets are created.

DAY,WEEK,MONTH, QUARTER, YEAR adaptable
program

```
%macro sales(n=14,period=day);

data sales;
    set in.sales;
    saledate = intnx("&period",saledate,0);
    if saledate ge intnx("&period",today(), - &n);
run;
%mend sales;
```

Of course, this is a simple macro, but it demonstrates how macros can make building systems and writing applications much easier.

KEEPVAR Macro

The next example shows a macro tool that is used to keep variables in a data set only if they are also in another "base" data set. This can be useful when a data set is being prepared to replace a permanent data set. The new data set may have additional variables on it that were used to create the data set but are not required to be on the data set when it is stored. This macro is a tool because it performs a very general function and can be used in any system.

```
%MACRO KEEPVAR(DATA=, BASE=, OUT=,
                PREFIX=);
```

```
%LET LONG = %LENGTH(&PREFIX);
%IF &LONG GT 5 %THEN %LET LONG = 5;
```

```
%IF &LONG GT 0 %THEN
    %LET PREFIX = %SUBSTR(&PREFIX,1,&LONG);
```

```
PROC CONTENTS DATA = &BASE NOPRINT
OUT=&PREFIX.VAR;
RUN;
```

```
DATA _NULL_;
DO UNTIL (EOF);
    SET &PREFIX.VAR END=EOF;
    COUNT + 1;
    CALL SYMPUT ('V'||LEFT (PUT(COUNT,5.)),
NAME);
END;
```

```
CALL SYMPUT ('NUMVARS',
LEFT(PUT(COUNT,5.)));
RUN;
```

```
DATA &OUT;
    IF 0 THEN SET &BASE; /*GET VARIABLE
LENGTHS*/
    SET &DATA;
    KEEP %DO I = 1 %TO &NUMVARS;
        &&V&I
    %END;
;
```

```
PROC DATASETS NOFS;
    DELETE &PREFIX.VAR;
RUN;
```

```
%MEND KEEPVAR;
%KEEPVAR(DATA=Y, BASE=X, OUT=Z,
PREFIX=ABCDEFGH)
```

This macro shows that a piece of SAS code which is fairly complicated, but it performs a conceptually simple task. Because the code is part of an easy to use macro, inexperienced programmers can take

advantage of programs written by someone else. Of course, even experienced programmers would want to use the macro because it is easier than rewriting the code and their application programs will be less cluttered.

The *advantages of macro-based systems* over standard SAS code are clear: systems are implemented more quickly, programs within a system are more consistent, and code can be executed conditionally depending upon input data, parameters, or global macro variables.

Standards

Once you have decided to build macro-based systems, you must resolve some issues with regard to your own organization in order to make your systems easy to build and maintain. In a word, you need *standards*. Standards are a set of rules, such as coding conventions, naming conventions, and documentation requirements, which help to make computerized systems easier to build and maintain. There is one key point that should be remembered when implementing standards: they should make your job easier - not harder.

Here are some characteristics of standards that you should keep in mind:

- Standards are *management tools* - not commandments meant to last forever.
- Standards should *help you* get systems built faster and help you to maintain them. They should not inhibit system construction needlessly or make system maintenance more difficult.
- *Automated techniques* should be employed whenever possible to help programmers follow standards and monitor their adherence to standards. This makes the programmer's job easier and allows management by exception to insure compliance with standards.
- Standards should be *useful for everyone* - from beginning programmers to experienced programmers familiar with your particular systems. This means that they will probably have to be able to bend to prevent them from breaking. They must be flexible and useful at the same time.
- A characteristic of bad standards is that they become more difficult to adhere to as your programming and system development skills

improve. *Bad standards must be changed*, or they will degrade programmer efficiency.

- Since standards are management tools, it is usually a good idea to use characteristics of *your organization's management structure* as a basis for some of your standards.

Keeping those general characteristics of standards in mind, there are some issues that should be examined with regard to setting up macro-based systems.

Primary Concerns

The primary concerns are as follows:

- A central macro library versus decentralized macro libraries.
- Should naming conventions encourage descriptive names such as %LATE or non-descriptive names such as %L0001 or some combination of the two.
- Global macro variables - which ones should there be and what should their values mean.
- How are your macros and macro variables documented? How is this information disseminated throughout your organization?

A summary of the various trade-offs concerning standards you will have to consider is shown below. Some suggestions about the best approach to use are also provided.

Centralized Macro Libraries

Advantages:

- All macros are in one place
- No two macros can have the same name because a Partitioned Data Set (PDS) can only have one member with a particular name
- Macros can be centrally coordinated with standards centrally enforced

Disadvantages:

- More difficult to test macros. Temporary macro libraries must be used or macros must be explicitly included.
- Getting central approval for macros can delay project completion

- Sheer volume may be overwhelming for the person administering the central macro library. This can lead to either project delays or standards being poorly enforced or some combination of the two.

Decentralized Libraries

Advantages:

- Lets control of libraries be patterned after control of organization
- Smaller libraries are more manageable. You can have each of the following type of libraries so that the macros in them can be managed at the appropriate level:
 - * **Personal** - Used for personal macros. Typically used by a programmer to set up a display manager environment, test new macros, and keep personal productivity macros.
 - * **Project** - Used to store macros used for a particular project and to test macros being developed for that project. These are typically application macros for writing reports in particular formats, subsetting observations, and creating macro variables for that particular project.
 - * **Department** - Used to store macros invoked in a variety of projects in a department. Such macros will mostly be application macros, but there will usually be some macro tools also. The line between application and tool macros sometimes gets blurred in these libraries. (One programmer's tool is another programmer's application.)
 - * **Company** - Used primarily for storing macro tools and macros that initialize company level global macro variables. Also, any macros used by more than one department should be stored in this library.
- Allows easy testing of macros - macros can be easily put into personal or project libraries for testing and then moved up to a department or the company library if appropriate.
- Projects are not delayed because a central authority must approve every macro - macros can be put into project macro libraries immediately and be moved to the company library at a later time.

Disadvantages:

- Documentation and standardization might suffer - not requiring programmers to have their macros checked before having them put in a library permits them to ignore standards.
- Two macros with the same name could exist in two different macro libraries. In some situations, this could lead to confusion.
- There could be duplication of effort within a company. One project team could be working on a macro that another project team has already developed.
- There will be more data sets (libraries) to keep track of. Instead of one large, central PDS, there will be dozens or hundreds of smaller PDS's.

How MACRO LIBRARIES are accessed in an OS Batch Environment:

CENTRALIZED

```
//SASAUTOS DD DSN=MAINLIB,DISP=SHR
```

DECENTRALIZED

```
// SASAUTOS DD DSN=MYLIB,DISP=SHR
//          DD DSN=PROJLIB,DISP=SHR
//          DD DSN=DEPTLIB,DISP=SHR
//          DD DSN=COMPLIB,DISP=SHR
```

The conclusion: Decentralized libraries are usually better because they allow for more flexibility, easier management of any one particular library, and they make it easier for people to share macros. Ironically, sharing macros is easier because people have less macros to look through in order to find existing macros that will serve their purposes. The company and department macro libraries will be smaller than one central company library would be. As an added bonus, the control of macro libraries can be patterned after the management structure of the company.

Using a Centralized macro library may be appropriate for installations concerned about the proliferation of partitioned data sets or for

organizations that have relatively few people writing macros.

NAMING

Descriptive Names

Advantages:

- Easy to understand
- Simple

Disadvantages:

- 8 characters might not allow a good description
- Similar macros or variables will be hard to distinguish

Examples: REPORT, RPT, REPORTER

They all have something to do with reporting, but how are they different?

Non-Descriptive Names

Advantages:

- Names are not misleading.
- No imagination is needed for naming macros and variables. This can be a real advantage if you have to name dozens of macros and macro variables.

Disadvantages:

- Explicit documentation is absolutely essential. Since the name contains no information, the documentation must tell the programmer what the names mean.
- Less Intuitive - It is difficult to guess the purpose of a macro or variable from its name.

Examples: MAC001, MAC002, UTL101

Except possibly for three letters at the beginning of the name, these macro or variable names provide no clue as to their purpose.

The conclusion: A combination of the two methods should be used. Descriptive names should be used for high level activities such as certain global macro variables, macro tools, and certain application macros. Most application macros should have a prefix that identifies them as belonging to a particular phase of a system with a number or letter system for the rest of the name. Macro parameter names should, if possible, always be descriptive. Most macro variables should have descriptive names. A general rule of thumb is that descriptive names should be used up to the point

when your systems become too cluttered to use them effectively.

GLOBAL VARIABLES

- For use in all Macros - any macro that does not refer a local variable of the same name can access the global macro variable.
- Good with SAS/AF® - Eliminates confusion caused by AF screens being invoked in different macro referencing environments.

GLOBAL VARIABLES vs. PARAMETERS

Global Variable:

- Any macro can change value
- Should be defined as GLOBAL in every macro
- Can require slightly less coding (for parameters).

Parameters:

- Individual macros can be tested
- Modularity

GLOBAL VARIABLE EXAMPLES

Variables You Might Want to use

ERRORMSG	An error message
ABORT	YES or NO
DEBUG	YES or NO
FIRSTOBS	a Number
OBS	MAX or a Number
TEST	YES or NO

The conclusion: Macro variables such as DEBUG and ABORT can be useful as global variables because you want them to affect every macro in your system. You may also want any of your

macros to be able to change them in the event a processing problem occurs.

Parameters can often be very useful in situations where macros are transported from one library to another where they are exposed to different global macro variables. The modularity created by using parameters makes the macro more portable. They also enable the programmer to better control or eliminate side effects from macro invocations.

Advertising Macros

Like with many businesses, the most effective advertising for macros is word of mouth. One programmer tells another programmer about a macro that is useful. As effective as that is, it is a fairly hit and miss process. For macro-based systems to be fully taken advantage of, there has to be a more effective way of disseminating information about macro building blocks - applications and tools. Certain techniques tend to be fairly helpful.

First of all, each macro must have good documentation describing its purpose and its parameters. That makes it easier for someone to use an existing macro rather than build a new one because they are sure what the existing macro does. It is helpful to have documentation on-line as well as printed.

Next, it helps to have decentralized macro libraries. Smaller libraries make it possible for management at appropriate levels to direct the attention of their subordinates to particular macros that may be of interest. There is a better chance that someone in charge of a small library devoted to some particular area (such as a department or project) will know their macro library better than someone who administers an enormous, company macro library.

And it is also useful to have design meetings and code walk-throughs. There is a better chance that at least one member of a group of people knows more than one individual about a macro that can effectively perform a task.

GOOD TIPS for building Successful macro-based systems:

- Explicitly define every macro variable in a macro to be LOCAL or GLOBAL with a %LOCAL or %GLOBAL statement unless it is a parameter. Parameters are, by definition, LOCAL macro variables.

- Use Keyword parameters rather than positional parameters. They can be set with default values, the order they appear in does not affect how they work, and it is easier to understand what the program invoking the macro is doing.
- Use parameters as much as is practically possible - This keeps your macros more adaptable and easy to use when minor adjustments are needed. For example, input and output data set names should be able to be changed by using a keyword parameter.
- Incorporate error checking into macros - Give yourself or the maintenance programmer some helpful messages in case something goes wrong. Also, prevent disasters by cross checking parameter values, when appropriate, to see if they make sense in relation to one another.
- Use the RUN statement to end DATA and PROC steps. This will save you a lot of grief. Not using a RUN statement can cause a great deal of confusion about macro referencing environments and when data and proc steps get executed in relation to macro statements.
- Use common sense and expert advice when setting up standards. Remember that standards are supposed to help programmers, not decrease their efficiency. Also, if you do not know what your standards should be or are unsure of how they should change in the future, discuss your situation with a more experienced person who is able to give you sound advice.

Conclusion

The main points to consider about macro-based systems are they are more easily managed and maintained than "loose" SAS code. You are working with building blocks instead of SAS statements. The building blocks you construct provide for standardization among different application programs. They also enable you to build new systems quickly. Good standards and administration of your systems can help to make them more effective, but bad standards will damage productivity. *Good management and administrative skills are just as important as technical skills when it comes to managing macro-based systems.*

References

From SAS Institute Inc.:

SAS Guide to Macro Processing
SAS Users Guide Basics
Technical Report P-146

SUGI 13 Proceedings:
Peter Kretzman, "Teflon and Tang: Forging Generic
Macro Tools"

SAS, SAS/AF, and SAS/FSP are registered
trademarks of SAS Institute Inc., Cary, NC

If you have any questions or comments, feel free
to contact the author at

Levine Software Systems
5525 Queen Avenue South
Minneapolis, MN 55410
(612) 3926-6967