

When %DO I Use Macros and When %DO I Not

by

Allan M. Kominck

ABSTRACT

The author has been using macros for over 15 years and at one point was somewhat obsessed with them. The depth of this obsession came to light when he attempted to name his children %Katie and &Courtney. After many hours of soul searching and the one incident with the %LET statement, he is feeling much better now. This presentation will deal with the appropriateness of using macros. When does the power and efficiency of the macro language outweigh the complexity that can be added to program code? The decision to use macro code is strictly up to the programmer. The goal of the presentation is to provide information for making that decision.

INTRODUCTION

This could very well read like the plot of one of those low budget sci-fi movies from the fifties during the dawn of the atomic age. The young scientist (programmer) discovers the new subatomic particles (SAS® macro language). The & particle and the % particle. He has the power of the universe at his finger tips. Will he use it for good or for evil? Will he be able to stand the pressure or will he be crushed by it? The young scientist may perish in the movie, but truth, justice, and the American way usually triumph. But our young programmer may end up with systems that eat the CPU alive! The message of our little vignette is that powerful things just can't be unleashed. They must be managed and controlled. Management and control require planning and forethought. Let's look at the power and then consider some of the things that could go wrong.

THE POWER AND THE GLORY

I can remember when I was just getting into SAS programming and I thought I was hot stuff. There was absolutely nothing that I could not do with SAS. Like all young and foolish people I

soon found that I was having trouble making my SAS programs do certain things. Things like executing a DATA step only if certain conditions were true. Getting conditional type information into TITLE statements without creating all sorts of DATA set variables. I found out all too quickly that I wasn't such hot stuff after all. I even entertained thoughts that the fault might be with the SAS System. I don't remember exactly how it happened. Whether someone told me or I saw someone else's code or if I stumbled onto it in a manual or some publication, but I found out about macro code. (I definitely was not staring through an electron microscope or looking at the results of my last atom smasher run.) Although the original shortcoming was mine, I had found a way to be hot stuff again. I could use macro code to conditionally execute DATA and PROC steps. I could add text to TITLE statements based on certain conditions. I could even allow the SAS System to write repetitious code for me. I was unstoppable. Mine eyes had seen the glory! I began slowly. An & here, a % there. It was exhilarating. My colleagues were amazed. The more praise they heaped on me, the harder I strove to include macro code into every program that I wrote. What followed was not pretty. Words like elegant and inspired were no longer good enough. I wanted to soar like Icarus and take my place on Mount Olympus. At first people could not understand my coding. In my mind that only seemed logical because I was unique, one of a kind. Eventually, people could not understand me, period. I was an enigma. I needed help. After much soul searching and a great deal of help and understanding from my family and colleagues, I was able to take my place among "normal" SAS programmers, whatever that may mean.

THE HAPPY MEDIAN

There is a fine line that separates the judicious use of macro code from the fanatical. Somewhere in the misty past, I'd crossed that line. The difficulty for me lies in drawing that line. To me the boundary is ill-defined, often varying in width and shifting positions. The line attempts to mark the point where the costs outweigh the benefits. In some instances, it may be very difficult to place an exact value on either these costs or benefits, most programmers can place a guesstimate on them. Each individual guesstimate will vary slightly based on the individual's perception and the individual's acceptance of risk. This variation explains the shiftiness of the line. With this disclaimer under our belts, let us proceed to consider the different sides of the line.

THE BENEFITS

A lot of good things come from the use of macro code.

Reusable code.

We've all had that one program that we just keep cloning or modifying to meet various people's needs. The classic example is a program with IF statements like the following:

```
IF DEPT="ACCT" THEN ... ;
```

Produces a report for the Accounting Department. Well, soon the engineers want the same report and the program gets copied and all the IF statements get changed:

```
IF DEPT="ENG" THEN ... ;
```

Soon were running out of clever names to give these programs because we've copied them so much. Another approach using macros would be to write one program with a %LET that defines a macro variable and then use the macro variable in all the IF statements:

```
%LET DEPT=ACCT;
```

```
IF DEPT="&DEPT" THEN ... ;
```

To run this same report for the engineers, all we have to do is change the %LET statement:

```
%LET DEPT=ENG;
```

and run the program again to produce the new report. Another approach would be to make the entire program a macro.

```
%MACRO ANYRPT(DEPT);  
IF DEPT="&DEPT" THEN ... ;  
%MEND;
```

The %LET statement is no longer needed as the value for DEPT is passed to the macro when it is called. To run the report for the accountant:
%ANYRPT(ACCT);

To run the report for the engineers:

```
%ANYRPT(ENG);
```

If we are clever enough, we can identify tasks that we often find ourselves recoding or copying over and place them into a library of utility programs. For example, if we write a really slick routine for performing word searches on large text strings and there is a great need for such a routine, it can be converted into a macro that can be called by other programs when needed.

Conditionally executed DATA & PROC steps.

Those of us who grew up using SAS in a batch environment on the mainframe have all experience the following scenario. The program that we've just written is fairly complicated and it reads some pretty hefty data bases. The job runs for hours and there is no output. Was there an error that prohibited the job from going to completion or were we smart enough to have chosen the right criteria to have selected zero observations. Ooops, we didn't think to send the log to the hold queue! What's a programmer to do? One approach that I've taken in programs that allow the user some discretion in selecting the data that will be printed is to check the final data set to make sure that there is data in it. If the data set is empty, I print a report that states that no data was selected based upon the user's criteria.

This is accomplished using macros and a CALL EXECUTE statement. Let's take a look at an example, and I'll explain how it works.

```
DATA DUMMY;  
DEPT="ZZZZ";
```

```
DATA _NULL_;
```

```
SET DUMMY FINAL END=EOF;
IF _N_ AND EOF THEN
  CALL EXECUTE(%PROBLEM);
ELSE CALL EXECUTE(%CONTINUE);
```

A dummy data set called, of all things, DUMMY is created. DUMMY contains one observation, the fictitious department ZZZZ.

The data set FINAL is the data set from which we wish to generate a report. All subsetting of data has taken place at this point in the program and we are just curious if there is indeed any data in the data set. The FINAL data set is concatenated to the DUMMY data set. The IF statement in the data step checks if there is only one observation after the two data sets are concatenated. If there is only one observation, it came from the DUMMY data set and the FINAL data set is empty. If this is true, the PROBLEM macro is run. The PROBLEM macro could be something like the following:

```
%MACRO PROBLEM;
DATA _NULL_;
  FILE PRINT;
  PUT @1 'No Observations found for selection
criteria';
  PUT @1 'No Report can be produced.';
%MEND;
```

If on the other hand, the first observation is not the end of the file, the CONTINUE macro is run. The CONTINUE macro would contain the code that would normally be used to print the report. The beauty of the CALL EXECUTE statement is that the macro invoked can contain any type of SAS coding including DATA and PROC steps. The DATA and PROC steps will not execute until the step in which the CALL was made has finished executing. That is why a simple macro was not used, because SAS would attempt to execute the resultant code from that macro during the execution of the DATA step that invoked it.

Performing Repetitive Tasks

Remember our discussion of reusable code? We talked about cloning programs and just changing them slightly. We've also had instances where we've found ourselves duplicating a section of code in a program, modifying it slightly and then repeating the process. For example, you have to print a series of reports by department. No big

deal, use a by statement! Well, there is a catch. Each report has to go to a different print destination. Here's the usual scenario. Get the code for the first report all set, then copy the code and change the selection criteria and the print destination.

```
DATA _NULL_; SET ALLDEPTS;
  IF DEPT="ACCT";
  FILE OUT1;
  PUT ...
```

Copy the code change ACCT to ENG and OUT1 to OUT2. Repeat as needed.

By using macro code, we can have SAS do the work for us.

```
%LET DEPT=DEPT;
%LET DEPT1=ACCT;
%LET DEPT2=ENG;
%LET DEPT3=SALES;
```

```
%MACRO DEPTRPTS;
%DO I=1 %TO 3;
  DATA _NULL_; SET ALLDEPTS;
  IF DEPT="&&DEPT&I";
  FILE OUT&I;
  PUT ...
%END;
%MEND;
```

The %DO will execute 3 times and create three DATA _NULL_ steps that will accomplish the same result as copying the code and modifying it. Note the double ampersand (&&) in front of DEPT&I. The macro processor will take multiple passes in order to resolve macro variables. On the first pass, &DEPT will resolve to DEPT and &I will resolve to 1, leaving &DEPT1. On the second pass &DEPT1 will resolve to ACCT leaving the following SAS statement:

```
IF DEPT="ACCT";
```

These are just a few examples of the benefit of using SAS macros.

Now to the Dark Side!

THE COSTS

We must be careful in our use of macros, lest we pay the price. The price may take the form of increased CPU usage or increased work loads for ourselves. How could an innocent macro cause us to overtax our computing resources? Let's look at our example of the %DO.

```
%MACRO DEPTRPTS;
%DO I=1 %TO 3;
  DATA _NULL_ ; SET ALLDEPTS;
    IF DEPT="&&DEPT&I";
    FILE OUT&I;
    PUT ...
%END;
%MEND;
```

This macro will create three DATA _NULL_ steps, each of which will generate a report for a separate department from the SAS data set ALLDEPTS. What if we modified the code slightly?

```
%MACRO DEPTRPTS;
%DO I=1 %TO 3;
  DATA _NULL_ ;
    INFILE "ALLDEPTS.DAT";
    INPUT ...;
    IF DEPT="&&DEPT&I";
    FILE OUT&I;
    PUT ...
%END;
%MEND;
```

What's changed? We are no longer setting a SAS data set, but reading in the data from a sequential data set, ALLDEPTS.DAT. No big deal, right? Well, let's look at what the macro is going to do now.

Again, three DATA _NULL_ steps are going to be generated, but each time it is going to be generated from what we are assuming to be the raw data file. This raw data file may contain millions of records. It may contain data on other departments other than the three that we are interested in. It does not appear that we are using keys to access the data, so we are going to read all of the records then only keep those for DEPT="ACCT". Then we are going to read all the records again and only keep those for

DEPT="ENG". This is definitely one of those times when we are going to pay for using a macro.

The other way that we may pay the price is through increased work loads. How can a macro increase our work load? As developers, it is saving us so much time by allowing us to create reusable modules, to conditionally execute DATA and PROC steps, and to perform repetitive tasks. How could doing all this for us possibly increase our work loads? If we are the ones who use the code and maintain it, yes, macros will save us time. But what about those situations where an application is developed for a user or group of users and the application is turned over to the users to maintain? If the users have the same level of expertise in SAS as the developer, there should be no problem. But if they had the same level of expertise as the developer, why did they have the developer code the application in the first place? Consider the people who will have to deal with the code. For instance, it may save us all kinds of time in the future if we hard code three data steps rather than create them with a %DO. All SAS users understand DATA steps, not all SAS users understand macro code. If our users do not understand macros, we are definitely going to get a phone call about our application, the first time our users try to add another report to the system. Whereas, they may have been very comfortable, copying code and modifying it. Trust me on this one, I think I set a record for the most help calls in one day during my macro phase.

CONCLUSION

As we have seen in our brief discussion, macros are a powerful tool that can save time when developing applications and that can add flexibility to applications. But like any tool, they must be used properly and in the right circumstances. In ancient Greece, those wishing to approach the oracle of Delphi were first cautioned to know thyself. In this case, we must not only know ourselves, but know our users and know our macros!

SAS and SAS/AF are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.