COMMON MISUNDERSTANDINGS
ABOUT THE SAS® MACRO FACILITY
Earl Westerlund, Eastman Kodak Company

ABSTRACT

The macro facility in base SAS® software is a powerful tool which can increase the functionality and flexibility of SAS programs. The macro language is also very complex, and misunderstandings about how the language works can cause problems for beginning users.

This tutorial will discuss a few of the common problems that beginners (and experienced users) have with the SAS macro facility. Intuitive, plain-English answers will be given to some questions that are frequently asked.

INTRODUCTION

I work in the Information Resource Center at Kodak Park. Kodak Park is the largest site of Eastman Kodak Company, employing over 25,000 people. In 1987, the IRC received over 2,600 calls about the SAS system. Our customers' experience levels range from novice to expert. One part of the SAS system which trips up users in the intermediate to advanced range is macros. Macros are very useful but can be very confusing. Their strength (power) goes hand-in-hand with their weakness (complexity).

This tutorial will go over eight common questions which we are asked frequently. We will talk a little about the problem behind the question, then concentrate on the solution to the problem. The explanations of the problems and solutions will be intuitive. The use of terminology will not be rigorous.

Two topics which will not be covered are quoting functions (a tutorial in itself) and the relationship of macros to SAS/AF® variables.

All topics to be discussed are covered in the "SAS Guide to Macro Processing" and the "SAS Macro Language" training course. Those attending this tutorial should have some experience writing simple SAS macros, but should not necessarily be experts at writing SAS macros.

QUESTION:

"I %INCLUDE-d my macro. Why didn't it run?"

PROBLEM:

This question comes most often from people who are used to including open SAS code from external files. The basic problem is that "including" just reads the SAS code from the file, and only defines the macro. Simply defining a macro does not execute it. The macro must be invoked as well.

SOLUTION #1:

Consider whether you need a macro at all. If there is no conditional code to be executed, perhaps simply storing the SAS code in a file and including it when needed is all that is necessary:

%INCLUDE MYCODE;

SOLUTION #2:

If you do need a macro, simply invoke the macro after including it:

%MYPAC;

SOLUTION #3:

If you have a number of macros, you may want to take advantage of the AUTOCALL option of SAS. Put all your macros in a macro library (a MACLIB on VM, a partitioned data set on MVS), then define the library as an "autocall" library when invoking the SAS system. When you need to, simply invoke the macro:

%MYPAC

The AUTOCALL feature will read the macro from the library, if necessary, before invoking it. Contact the SAS consultants at your site for information on using the AUTOCALL feature.
QUESTION:
"I defined my macro variable. Why didn’t the program recognize it?"

PROBLEM:
The macro variable was probably defined in a macro. Macro variables that are created within a macro exist only while that macro is executing. When the macro is done executing, the variable disappears.

SOLUTION:
Use the %GLOBAL statement in the macro to tell SAS to keep the macro variable around after the macro is done executing. If your macro variable is &X, then the statement would be:

%GLOBAL X;

QUESTION:
"I created a macro variable using CALL SYMPUT. Why didn’t the program recognize it?"

PROBLEM #1:
A macro variable which is created using SYMPUT is not available to be used until AFTER the DATA step is done. There are two common problems here. The first is when you actually want to use the macro variable right away. For example:

DATA _NULL_;  
SET SAVED,DATA;  
CALL SYMPUT ('VNAME',VNAME);  
TITLE "Report for &VNAME";

This will not work, because the macro variable &VNAME will not be defined until after the DATA step has been run.

SOLUTION #1:
Break the DATA step into two DATA steps – the first to create the macro variables, the second to use them:

DATA _NULL_;  
SET SAVED,DATA;  
CALL SYMPUT ('VNAME',VNAME);  
STOP;  
RUN;  
DATA SUBSET;  
SET SAVED,DATA;  
IF &VNAME GT 1;  
RUN;

PROBLEM #2:
This problem occurs when you intend to use the macro variable after the DATA step executes, but the lack of a RUN statement means that it has not executed yet. For example:

DATA _NULL_;  
SET SAVED,DATA;  
CALL SYMPUT ('VNAME',VNAME);  
TITLE "Report for &VNAME";

The TITLE statement executes immediately. It does not wait until the DATA step has executed, so &VNAME has not been defined yet.

SOLUTION #2:
For the second case, simply add a RUN statement at the end of your DATA step, to force it to execute before the TITLE statement:

DATA _NULL_;  
SET SAVED,DATA;  
CALL SYMPUT ('VNAME',VNAME);  
RUN;  
TITLE "Report for &VNAME";
QUESTION:
"Why don't macro variables work in TITLE statements?"

PROBLEM:
Most likely the title text is in single quotes (''). The SAS system takes everything within single quotes exactly as given, without attempting to resolve the macro variable.

SOLUTION:
Make sure the DQUOTE option is on, and include the text of the title in double quotes ("). For example, if the macro variable &PRODUCT has the value XYZ, and the TITLE statement reads

```
TITLE 'Product &PRODUCT';
```

The title on the listing will be

```
Product &PRODUCT
```

If the statements are

```
OPTIONS DQUOTE;
TITLE "Product &PRODUCT";
```

Then the title on the listing will be

```
Product XYZ
```

QUESTION:
"The data set name changes each time the program is run. Can I get the program to ask the user for it?"

SOLUTION:
You can use the %PUT and %INPUT statements to get a program to talk to the user. For example:

```
%PUT Enter data set name::;
%INPUT DSNNAME;
PROC PRINT DATA=&DSNAME;
```

%PUT writes the text to the log file, so your program must either run interactively on TSO or CMS, or use the LT option if you are running noninteractively under CMS.

QUESTION:
"I can figure out that 1+1=2. Why can't the macro language?"

PROBLEM:
The macro language is a character processing language. As far as it is concerned, numbers are characters like every other character.

SOLUTION:
The function %EVAL can be used to evaluate integer numeric formulas. So, while the statements

```
%LET Y=1+1;
%PUT The answer is & Y;
```

produce

```
The answer is 1 + 1
```

the statements

```
%LET Y=%EVAL(1+1);
%PUT The answer is & Y;
```

produce

```
The answer is 1+1
```

Unfortunately, %EVAL only does integer arithmetic.
**QUESTION:**

"I'd like to execute two different macros depending on the value of a variable. But when I put this code in:

```
IF X = 1 THEN %MACROA;
ELSE %MACROB;
```

the program gets confused. Why?"

**PROBLEM:**

The SAS system takes several passes to put together and run a DATA or PROC step. First, if there are any macros or macro variables they are resolved. Then the resulting DATA or PROC step is compiled and run. That means that the above macros will have been turned into SAS statements by the time the program is run. If the macros do not begin with a DO statement and end with an END statement, or contain other DATA or PROC steps, you will probably get a syntax error.

**SOLUTION #1:**

Assuming that the macros contain complete SAS program statements, you could do the following:

```
IF X = 1
THEN DO;
  %MACROA
END;
ELSE DO;
  %MACROB
END;
```

**SOLUTION #2:**

Another possibility is to use the CALL EXECUTE statement:

```
IF X = 1
THEN CALL EXECUTE ("%MACROA");
ELSE CALL EXECUTE ("%MACROB");
```

There are two things to be aware of when using CALL EXECUTE:

- The macro which is invoked by CALL EXECUTE is executed immediately, but any SAS program statements generated by the macro are executed AFTER the DATA step is done.
- Because a DATA step executes once for every observation in the data set you are reading, the macro could conceivably be called that often. If you are not careful, you could invoke the macro hundreds or thousands of times without realizing it!

**QUESTION:**

"I used a %INPUT statement to read the name of the type of plot the user wants, but the program never executes the right plotting macro. Here's the code:

```
%INPUT PLOTTYPE;
%IF &PLOTTYPE = ABC
%THEN %PLOTABC;
```

Even when they enter "abc", they don't get the plots. Are you sure that this PUT/INPUT stuff really works?"

**PROBLEM:**

I'm sure it works. The problem is likely that the user is entering the answer in lower case. If the character string "abc" is compared to character string "ABC", the result is that the comparison is unequal. By the way, the CAPS option does not affect macro processing.

**SOLUTION:**

The %UPCASE function will translate all letters in a macro variable to uppercase, so that comparisons can be made:

```
%IF %UPCASE(&PLOTTYPE)=ABC
%THEN %PLOTABC;
```
CONCLUSION:

The SAS macro language is a very useful and powerful tool. However, it is so complex that even an experienced user can fall into traps. This overview discussed some of the common problems people have with the SAS macro facility, and solutions to those problems.

Many misunderstandings that occur happen because users don't know "what gets done when" and "what gets stored where". Here are some resources that are available to gain a better understanding of the macro facility:

- "Using SAS® Macro Variables" is a videotape from SAS Institute which covers the basics of using macro variables. It has a particularly good explanation of how SAS statements are processed.

- The "SAS® Guide to Macro Processing" is dedicated to the macro facility. It contains explanations and diagrams which facilitate understanding.

- The "SAS® Macro Language" training course, offered by the SAS Institute Training Department, is a comprehensive overview of the macro facility.

REFERENCES

- Technical Report P-146: Changes and Enhancements to the Version 5 SAS System

SAS and SAS/AF are registered trademarks of SAS Institute Inc., Cary, NC, USA. No endorsement by Eastman Kodak Company is implied.