A SYSTEMATIC APPROACH TO THE UNDERSTANDING OF SAS MACROS

Bernie Siryk
WANG Laboratories, Inc.

ABSTRACT

A systematic approach to understanding the new macro facility is presented. Included in this discussion will be proper definition and calling of a macro, identifying each of these tasks, examining the context in which a macro is called, proper use and passing of macro variables, and a general overview of using macro programming statements. Several common errors regarding macro language usage accompanied by a brief discussion of macro options and how they may be used for debugging purposes will complete the presentation. Examples illustrating various points will appear throughout the discussion.

The 82.0 release of SAS included the introduction of an extensive, powerful macro facility. This new facility enables SAS users to define macros that accept parameters, retain macro variables across SAS steps, and conditionally create and/or execute SAS statements. Clearly, this new facility brings with it capabilities heretofore not available to SAS users. As with any major enhancement or feature, however, confusion and frustration tend to abound as the novice attempts to master the new technique(s).

In the author's own experience in mastering the new macro language as well as training/educating colleagues in using this new facility, a system of macro facility recognition and understanding evolved. The main thrust of this paper is two-fold: first, to present a viable system for recognizing and understanding the new macro language, and second, to discuss some of the more frequently committed errors using this new feature. A complete program containing a macro will be presented in order to illustrate the use of the macro facility.

Before presenting a systematic approach to understanding the macro facility it is important that the user understands the way SAS processes a program containing macro commands and/or variables. In order to avoid repeating in detail information presented on this very topic elsewhere (SAS Communications, Fall, 1983): Using the Macro Processor: What Happens When, pgs. 16-17), a brief summary will be presented. The SAS supervisor checks each incoming statement one at a time for syntax errors using a wordscanner. Statements are collected in this fashion until the current parser (being the DATA step compiler or PROC parser) encounters the end of the present step. Once an entire step is compiled, it is then executed (barring any syntax errors, of course). No more SAS statements are read until this process is completed.

The macro language processor is found between the wordscanner and the DATA step compiler or PROC parser. (See Figure 1, adapted from SAS Communications, Fall 1983, Pg. 16). The macro processor is triggered by "%'" and "&n" encountered by the wordscanner. At this point, the macro processor performs whatever action is required, be it resolution of a macro variable call and/or the execution of a macro programming statement, and puts the result on the input stack. From here, normal processing continues.

The first step in understanding macros is preceded by a fundamental rule: Before a macro may be used it must be defined. Therefore, the first step in looking at a program which contains macros is to identify where the macro is defined. After finding the macro definition, it is often helpful to scan the macro definition and take note of various things such as:

a) Macro Variables - Do the macros have any macro variables? If so, of what type? How are they created? Are they positional or keyword parameters? What default values are they set to?

b) Macro Test - What does it consist of? Are entire SAS steps outlined? Does the macro text consist of a SAS statement or two? Or is the macro text only a part of a SAS statement (i.e., a listing of variables which might be used on a VAR statement)?

c) General Issues - What is the main purpose of the macro? More often than not, the
macro name itself should present a hint as to the nature and/or purpose of that particular macro. If not, a perusal of the macro text should give the user a good idea of what it does.

The second main step in understanding macros stems from the basic rule that, once defined, a macro has to be "called" in order to use it. %MACRONAME is sufficient for calling a macro by that name. General things to look for at this point include:

a) Context in which the macro is called - Where in the program is the macro called? Is it called within a DATA step, within a PROC step, in several places? Moreover, given the general context and flow of statements around the macro call, how does our understanding of the macro (derived from steps above) fit in?

b) Parameter Values - What parameter values have been sent with the macro call? Conversely, what default values have been accepted? How will the parameter values effect the execution of the macro? If applicable, has a specific value been passed which conditionally alters the execution of the specific macro?

c) General Issues - What does the use of a macro at any particular point accomplish for the user? Is the macro required, and if so, is it being used efficiently? If two or more macros are nested within each other, will they accomplish their apparent purpose successfully? What sort of communication, if any, is found between the macros?

TRACING ERRORS WITH MACRO OPTIONS

Several macro options may be invoked which serve to assist the user in debugging SAS programs containing macros. Each option and its use is outlined below. Following the standard usage of all SAS options, these options may be specified in the program's JCL (Job Control Language) EXEC SAS card or within the program on the OPTIONS statement.

a) MACROGEN - This option prints the statements generated by the macro but does not show the resolution or substitution of macro variable values into the macro variable references. The NONPRINT option must be in effect in order to use MACROGEN.

b) SYMBOLGEN - This option produces a listing of the values for macro variables (referred to as symbolic variables) that are being substituted into the produced contrast to the MACRO statement? Was the macro definition ended appropriately using $MEND?

2. Correct usage of macro variables/parameters. Have they been referenced correctly using the "&" symbol? Has the distinction between positional and keyword parameters been adhered to in both the macro definition and the macro call? In passing the parameters around from one step to another, having referencing environment rules been violated (e.g., a local macro variable referenced globally)? Have key parameters not been assigned values in the macro call and/or assigned inappropriate values?

3. Correct usage of macro programming statements. Have macro programming statements been placed within a macro? Has the rule regarding "open" and "closed" macro statements been followed? Have macro programming statements been identified properly using the "W" symbol in order to distinguish them from standard programming statements?

COMMON PROBLEMS AND POINTS OF CONFUSION

Having learned the basics of the new macro processor, how it works and how to identify and understand what is going on, a watchful eye for some of the following pitfalls may well help the novice in avoiding and/or debugging macro problems:

1. Correct definition of macros. Was the syntax used for defining the macro correct? Was %MACRO used, in
text. As with MACROGEN, SYMBOLGEN requires that NOMPRINT be in effect.

c) MLOGIC - This option is used to illustrate macro processing logic. More specifically, MLOGIC indicates where branching and general flow within a macro occur. NOMPRINT must be in effect in order to use MLOGIC.

d) MPRINT - This option produces the SAS statements generated by the macro and prints them out in the order that SAS sees them. As noted above, MPRINT suppresses the execution of MACROGEN, SYMBOLGEN and MLOGIC.

---

Figure 1. Location of Macro Facility Within Overall Processing Scheme

---

129
NOTE: THE JOB DTBXSA HAS BEEN RUN UNDER RELEASE 82.3 OF SAS AT WANG DATA CENTER (01127001).

CPUID VERSION = 23 SERIAL = 021461

CPUID VERSION = 23 SERIAL = 221461

NOTE: SAS OPTIONS SPECIFIED ARE:

- MACRO SORT=4
- MACRO SYMBOLGEN NLISP ACLQUOTE NODEATE;
- DATA STATE;
- IF STATE=NLISPQUOTE("STR(S'ESTATES');
- TITLE PROC APROCNAME FOR STATE OF ESTATE;
- TITLE SUBMITTED ON &SYSDATE;
- SYMBOL GEN;
- DATA STATES;
- INPUT STATE $8. POP;
- CARDS;

NOTE: DATA SET WORK.STATES HAS 16 OBSERVATIONS AND 2 VARIABLES. 953 OBS/TRK.
NOTE: THE DATA STATEMENT USED 0.10 SECONDS AND 300K.

DATA STATEMENT USED 0.10 SECONDS AND 300K.

NOTE: DATA SET WORK.FLORIDA HAS 4 OBSERVATIONS AND 3 VARIABLES. 953 OBS/TRK.
NOTE: THE DATA STATEMENT USED 0.08 SECONDS AND 284K.

PROC MEANS FOR STATE OF FLORIDA SUBMITTED ON SYSDATE

VARIABLE N MEAN STANDARD DEVIATION MINIMUM VALUE MAXIMUM VALUE STD ERROR VALUE SUM VARIANCE C.V.
POP 4 1423484.75000 193183.445782 1234325.0000 1678900.0000 96594.122891 5693979.0000 37322161862 13.5711