STATEMENT STYLE MACROS

Donald J. Henderson, ORr, Inc.
David L. Kuhn, SAS Institute

1. INTRODUCTION

The SAS 82 MACRO facility can be used to define and add new statements to the SAS language. This tutorial will illustrate, by example, techniques for designing, testing and implementing statement style macros. A statement style macro is a macro whose call looks like a SAS statement. The call begins with a keyword, which is the macro name, and ends with a semicolon. The parameters follow the keyword and are separated by blanks. Particular emphasis will be given to techniques which allow the macro program to verify syntax before passing control to the SAS compiler.

This tutorial is for the advanced user who is thoroughly familiar with the SAS system and has experience with SAS 82 MACROs or TSO CLISTs or CMS EXECs.

The primary reason for using the STMT option is to tailor a SAS environment so that users and programmers can perform functions common to their environment using a syntax which they are familiar with. Thus, the STMT option can be used to define or build user languages. Given this use of the STMT option, it is crucial that meaningful error messages be issued if a mistake is made, just as the SAS system issues a meaningful message when a mistake in SAS syntax is made. In order to illustrate the building of "user-languages", a MAKEFMT macro will be defined initially. Next, the form of the call will be changed by using the statement option. MACRO code will then be added to edit the call of the MAKEFMT statement. Specifically, the following edits will be added:

- Verification that the specified input data set exists
- Verification that the names given for data sets, variables, etc. are valid
- Verification that the specified variables exist in the input data set.

The MACROs presented in this paper are for illustration purposes only. They are intended to serve as a guide to the MACRO developer/programmer in how to develop statement style MACROS which can edit their call. Furthermore, the techniques and MACROs presented here are also applicable to ordinary SAS 82 MACROS.

2. THE MAKEFMT MACRO

The data to be used in this example is illustrated in figure 1. ACT_ID contains coded values whose corresponding labels are the values of ACT_TITL. The values along with their labels are stored in a dataset rather than in a permanent format library in order to facilitate updating them as needed. The MAKEFMT MACRO will be developed to convert the information stored in this data set into a format module.

<table>
<thead>
<tr>
<th>ACT_ID</th>
<th>ACT_TITL</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>FINALIZE REQUIREMENTS</td>
</tr>
<tr>
<td>20</td>
<td>PREPARE FUNCTIONAL SPECS</td>
</tr>
<tr>
<td>30</td>
<td>REVIEW AND ACCEPT SPECS</td>
</tr>
<tr>
<td>40</td>
<td>DESIGN SYSTEM</td>
</tr>
<tr>
<td>50</td>
<td>CODE AND UNIT TEST</td>
</tr>
<tr>
<td>60</td>
<td>SYSTEM TEST</td>
</tr>
<tr>
<td>70</td>
<td>ACCEPTANCE TESTING</td>
</tr>
<tr>
<td>80</td>
<td>MOVE INTO PRODUCTION</td>
</tr>
</tbody>
</table>

FIGURE 1. Sample Data for MAKEFMT MACRO

The MAKEFMT MACRO has four parameters: the name of the data set containing the desired information (FROMDATA); the name of the variable with the code values (KEY); the name of the variable with the corresponding labels (RESULT); and the name of the format module to be created (NAME). As presently defined (see figure 2), the MACRO can be called with the following syntax:

```
%MAKEFMT(FROMDATA=PROJECTS,KEY=ACT_ID,RESULT=ACT_TITL,NAME=$ACFMT) |
```

The body of the MACRO has three major components. First is a data step program which reads the input data set and creates the macro variables FMT1 through FMTn, where n is the number of observations in the data set. The macro variable ENTRIES is given the value n, which represents the number of code values with labels which are contained in the input data set. The macro variables FMTI will have values of the form "code'='label," which will become part of the VALUE statement specification. Next, is the model or constant text which will be sent directly to the "$AS compiler. The third component is the %OO-%£ND block which is used to put the macro variables FMT1 through FMTn in the input stream. Last is the semicolon (;) ending the VALUE statement.

With these three major components, a SAS data set of codes and labels is read, parts of SAS statements are created and are then sent to the SAS compiler. This technique of building SAS code from information stored in a data set has applicability beyond the current example.

In order to make this into a statement style MACRO, /*STMT*/ is inserted before the
%MACRO MAKEFMT(FROMDATA=,KEY=,RESULT=,NAME=);
%* THIS MACRO CREATES A FORMAT MODULE FROM A GIVEN SAS DATA SET. IF A
%* THE DATASET IS EMPTY THEN A "NULL" FORMAT IS CREATED.
%*...
%RUN;
%DATA _NULL_; OPTIONS NONOTES;
SET &FROMDATA END=LASTREC;
&VAR='JMTTILEFT(PUT(''_',4.));
CALL SYMPUT('"",1"ITRIH(&KEY) II"I"TRIM(&RESULT) II"II";
IF LASTREC THEN CALL SYMPUT( '_ENTRIES', PUT('''_'),4.));
%RUN;
PROC FORMAT;
OPTIONS NOTES;
VALUE &NAME
%DO 1=1 TO &_ENTRIES;
&\_FIL&T&I
%END;
%MEND MAKEFMT;

FIGURE 2. The Initial MAKEFMT MACRO

1 OPTIONS IMPLYAC NOSOURCE2 TLS=72 LS=72 NODATE NONumber;
2 %INCLUDE MAKEFMT;
3
24 DATA PROJECTS;
25 INPUT ACT_ID $ ACT_TITL $40. ;
26 CARDS;
NOTE: DATA SET WORK. PROJECTS HAS 1 OBSERVATIONS AND 2 VARIABLES.
NOTE: THE DATA STATEMENT USED 0.36 SECONDS AND 576K.
35 MAKEFMT FROMDATA=PROJECTS KEY=ACT_ID RESULT=ACT_TITL NAME=ACTID;
NOTE: THE PROCEDURE FORMAT USED 0.52 SECONDS AND 640K.
124 PROC PRINT DATA=PROJECTS(DROP=ACT_TITL);
125 ID ACT_ID;
126 FORMAT ACT_ID $ACTID.;
127 TITLE SAMPLE PROJECT ID DATA WITH ACT_ID FORMATTED;
NOTE: THE PROCEDURE PRINT USED 0.20 SECONDS AND 640K
AND PRINTED PAGE 1.
128 RUN;
NOTE: SAS USED 640K MEMORY.

FIGURE 3. Sample Execution of MAKEFMT as a Statement Style MACRO

semicolon in the %MACRO statement. The form of the
 call now looks like a SAS statement and is
 illustrated in figure 3.

The next step in building the MAKEFMT
 statement is to test its results when called
 incorrectly. Specifically, call it with a data
 set name that does not exist. This is shown in
 figure 4. Note the number of error messages.
 A message that a data set was not found is
 given; however it is difficult to determine
 which one. Furthermore, error messages are
 generated for the PROC PRINT that executed
 successfully before.

3. EDITING THE MACRO CALL

The errors are due to the fact that the
 specified data set did not exist. Because of
 this, the macro variable ENTRIES was never
 created by the SYMPUT function. This caused
 several errors with the %DO-%END statement. In
 addition, this error then caused the text
 "PROC" to be interpreted as part of the VALUE
 statement, thus causing the remaining errors.
 The problem with ENTRIES can be easily fixed
 by initializing it to 0 using a %LET statement
 before the data step.

In order to fix the problem of the input
 data set which does not exist, a utility MACRO
 which checks to see if a data set exists can be
 used. This MACRO, given in figure 5, can then
 be called from within the MAKEFMT MACRO as
 shown in figure 6. If the specified data set
 has no variables and no observations it is
 presumed to not exist; CHECKDS prints a
 meaningful message to the log and sets a flag
 so that the MAKEFMT MACRO bypasses remaining
 processing. This is illustrated by the partial
 SAS LOG given in figure 7. Note also that the
 use of the text "ERROR:" in the %PUT statements
 causes the error to be reported at the end of
 the SAS LOG.
OPTIONS IMPLAC NOSOURCE2 TLS=72 LS=72 NODATE NONUMBER;
2 LINCLUDE MAKEFMT1;
2
24 DATA PROJECTS;
25 INPUT ACT_ID $ ACT_TITL $40.;
26 CARDS;

NOTE: DATA SET WORK.PROJECTS HAS 8 OBSERVATIONS AND 2 VARIABLES.
NOTE: THE DATA STATEMENT USED 0.37 SECONDS AND 576K.

ERROR: 200 COLUMN 2.
200: DATA SET NOT FOUND.

NOTE: $ACTID NOT OUTPUT DUE TO ERRORS.

35 MAKEFMT FROMDATA=PROJECT KEY=ACT_ID RESULT=ACT_TITL NAME=$ACTID;
66 PROC PRINT DATA=PROJECTS(IMPORTACT_TITL);

1

67 ID ACT_ID;
180
68 FORMAT ACT_ID $ACTID. ;

170
69 TITLE SAMPLE PROJECT ID DATA WITH ACT_ID FORMATTED;

ERROR: 1301 LINE 31 COLUMN 2.
170: FORMAT NAME IS NOT RECOGNIZED, OR MISSING $ SIGN.
PERHAPS A $ SIGN IS missing IN THE INPUT STATEMENT, OR YOU HAVE USED AN INFORMAT WHERE A FORMAT IS REQUIRED, OR VICE-VERSA.
180: STATEMENT IS NOT VALID OR IT IS USED OUT OF PROPER ORDER.
1301: APPARENT SYMBOLIC REFERENCE NOT RESOLVED.
1552: EXPECTED OPERAND NOT FOUND.
3: EXPECTING =.

NOTE: THE VARIABLE ACT ID IS UNINITIALIZED.
NOTE: SAS STOPPED PROCESSING THIS STEP BECAUSE OF ERRORS.
NOTE: THE PROCEDURE FORMAT USED 0.27 SECONDS AND 576K.

RUN;
NOTE: SAS USED 576K MEMORY.

FIGURE 4. Sample Execution of MAKEFMT With Nonexistent Input Data Set

%MACRO CHECKDS(DATASET);
%*---------------------------------------------------------*
%* THIS MACRO CHECKS WHETHER A GIVEN DATA SET EXISTS    *
%* DSGOOD SHOULD BE DECLARED LOCAL BY THE CALLING MACRO. *
%*---------------------------------------------------------*

DATA _NULL_; OPTIONS NODSNERR NONOTES;
IF _N_=0 THEN SET &DATASET;
ARRAY _N _NUMERIC;
ARRAY _C _CHARACTER;
DO OVER _C; CALL SYMPUT('DSGOOD', 'I'); STOP; END;
DO OVER _N; CALL SYMPUT('DSGOOD', 'I'); STOP; END;
IF LASTREC THEN CALL SYMPUT('DSGOOD', '0');
ELSE CALL SYMPUT('DSGOOD', '1');
STOP;
SET &DATASET END=LASTREC;
RUN;
OPTIONS DSNFERR;
%IF &DSGOOD=0 %THEN %PUT ERROR: DATASET &DATASET NOT FOUND OR IS NULL;
%MEND CHECKDS;

FIGURE 5. The CHECKDS MACRO
The bulk of CHECKDS' work is done by a special data step program. The NOSMFNDERR (no dataset found error) option is enabled. This causes references to nonexistent datasets to be interpreted as _NULL_ (i.e., zero observations and zero variables). The statement "IF _N_ = 0 THEN SET &DATASET;" serves a special purpose. It causes the variables in &DATASET to be defined to the program data vector (PDV) of the data step. However, it is important that the SET statement not be executed, thus the condition _N_ = 0, which can never be true. If the SET statement were executed and the data set empty or nonexistent, the data step would end before it had done its job. The ARRAY definitions use the special lists NUMERIC and CHARACTER to reference all the variables defined to the PDV at that point in the program. Programming logic is then used in the data step to assign a value to the macro variable DSGOOD. If there are any variables or observations it is set to true ('1'); only if there are no variables and no observations is it set to false ('0'). This macro illustrates that the capabilities and options of the SAS language itself, as opposed to the MACRO language, should not be overlooked in editing MACRO calls.

The CHECKDS MACRO only partially addressed the problem of an invalid data set because it does not edit for a valid data set name. When MAKEFMT is called with an invalid data set name as shown in figure 8, other unexplained errors are flagged in the SAS LOG.
In order to prevent these errors, another utility MACRO which checks names for validity can be used. This MACRO, CKNAME, is given in Figure 9. CKNAME is called by MAKEFMT as shown in Figure 10. This enhancement causes meaningful error messages to be issued as illustrated by the partial SAS LOG given in Figure 11.

The given version of CKNAME does not allow for two level dataset names. This could be easily added if desired. For purposes of this tutorial this limitation is not important. Comparable to the edit on the specified data set, MAKEFMT should check the specified variable names for validity. This can be done by adding code to MAKEFMT just before the call to CHECKDS, as illustrated in figure 12.

In addition to verifying that valid variable names are given, there is another level of editing that MAKEFMT needs to do. It should verify that the variables given for "KEY=" and "RESULT=" exist in the data set. A call of MAKEFMT with such an error results in another set of meaningless error messages in...
%MACRO MAKEFMT(FROMDATA=, KEY=, RESULT=, NAME=) /STMT;
%~':
%" THIS MACRO CREATES A FORMAT MODULE FROM A GIVEN SAS DATA SET. IF THE DATASET IS EMPTY THEN A "NULL" FORMAT IS CREATED.
%" 
RUN;
%LET ENTRIES=0;
%LOCAL DSGOOD DNAME;
%CKNAME(DATASET,&FROMDATA)
%IF DGOODNAME=0 %THEN
%DO;
%LET ERROR=FORMAT &NAME NOT CREATED;
%GOTO SKIPIT;
%END;
%CHECKDS(&FROMDATA)
%IF DSGOOD=0 %THEN
%DO;
%LET ERROR=FORMAT &NAME NOT CREATED;
%GOTO SKIPIT;
%END;
DATA _NULL_
OPTIONS NONOTES;
SET &FROMDATA END=LASTRECS;
MVAR='JITI'ILEFT(PUTCN_,4.));
CALL SYMPUT(MVAR.'· "JITI'I"II"II"II","II"II"II"II"II"II"II"II"II"II"II"II"II"II"II"II"II"II"II"II"II"II"II"II")
IF LASTRECS THEN CALL SYMPUTC'_ENTRIES',PUTI_N_.4.));
RUN;
PROC FORMAT;
OPTIONS NOTES;
VALUE &NAME
%DO 1=1 %TO &_ENTRIES;
&&_Fl'tT&I
%END;
%SKIPIT:
%MEND MAKEFMT;

FIGURE 10. MAKEFMT With Call to CKNAME to Verify Data Set Name

1 OPTIONS IMPMAC NOSOURCE TLS=72 LS=72 NODATE NONUMBER NOOVP;
2 %INCLUDE MAKEFMT3;
41 %INCLUDE CHECKDS;
65 %INCLUDE CKNAME;
96 DATA PROJECTS;
97 INPUT ACT_ID $ ACT_TITL &$40.;
98 CARDS;
NOTE: DATA SET WORK.PROJECTS HAS 8 OBSERVATIONS AND 2 VARIABLES.
NOTE: THE DATA STATEMENT USED 0.86 SECONDS AND 640K.
ERROR: LNGPROJECT IS NOT A VALID DATASET NAME
ERROR: FORHAT $ACTID Nor CREATED
107 MAKEFMT FROMDATA=LNGPROJECT KEY=ACT_ID RESULT=ACT_TITL NAME=$ACTID;
NOTE: SAS USED 640K MEMORY.
ERROR: ERRORS ON PAGES 1.

FIGURE 11. Sample Execution of Revised MAKEFMT With Nonexistent Input Data Set
%MACRO MAKEFMT(FROMDATA=,KEY=,RESULT=,NAME=)/STMT;
%* THIS MACRO CREATES A FORMAT MODULE FROM A GIVEN SAS DATA SET. IF
%* THE DATASET IS EMPTY THEN A 'NULL' FORMAT IS CREATED.
%*----------------------------------*-
RUN;
%LET _ENTRIES=0;
%LOCAL GOODNAME ERR1 ERR2;
%CKNAME (DATASET ,&FROMDATA)
%LET ERR1=&GOODNAME;
%CKNAME (VARIABLE ,&RESULT)
%LET ERR2=&GOODNAME;
%CKNAME (VARIABLE ,&KEY)
%IF &ERR1=O OR &ERR2=O OR &GOODNAME=O THEN
%DO;
%PUT ERROR: FORMAT &NAME NOT CREATED;
%GOTO SKIP1T;
%END;
%END;
%CHECKDS(&FROMDATA)
%IF &DSGOOD=O THEN
%DO;
%PUT ERROR: FORMAT &NAME NOT CREATED;
%GOTO SKIP1T;
%END;
DATA _NULL_
OPTIONS NOTEST:
SET &FROMDATA END=LASTREC;
MVAR=' _FMT'&LEFT(PUTLN_,4.));
CALL SYMPUT(MVAR, , , , , TRIM(&KEY) II '=' II TRIM(&RESULT) II ' , , ');
IF LASTREC THEN CALL SYMPUT('_ENTRIES' ,PCT(_N_,4.»);
RUN;
PROG FORMAT;
OPTIONS NOTEST;
VALUE &NAME
%DO 1=1 TO &_ENTRIES;
&&_FMT&I
%END;
%SKIP1T:
%MEND MAKEFMT;

FIGURE 12. MAKEFMT With Calls to CKNAME to Verify Variable and Data Set Names

The present version of CKVRS does not check whether variables are of the right type. However, this could be added using a variety of techniques.

A correct call of MAKEFMT is shown in figure 17. Note that all of the error checking has gone on "behind the scenes" with no indication in the SAS LOG.

MAKEFMT still does not handle all of the cases that it should. For example, this version of MAKEFMT still does not edit the format name. In addition it assumes that the input variables are character. These types of conditions can be edited for using an approach similar to the above. Another type of error is demonstrated in figure 18. Here, the MACRO call itself is invalid. The MACRO Facility assumes MAKEFMT has been called with invalid parameters. Note also that since blanks delimit the values passed in, it is not possible to pass a list in without enclosing it in quotation marks. Both of these problems can be solved by writing a "parser" callable from within the MACRO.
1 OPTIONS NLIN=AC NOSOURCE2 TLS=72 LS=72 NODATE NONUMBER NOOFP;
2 %INCLUDE MAKEFMT;
4 6 %INCLUDE CHECKDS;
70 %INCLUDE CKNAME;
101 DATA PROJECTS;
102 INPUT ACT ld $ ACT_TITL $ 640.;
103 CARDS;

NOTE: DATA SET WORK.PROJECTS HAS 8 OBSERVATIONS AND 2 VARIABLES.
NOTE: THE DATA STATEMENT USED 0.89 SECONDS AND 640K.

ERROR: 308 COLUMN 1.
ERROR: 307 COLUMN 34.
ERROR: 307 COLUMN 35.
ERROR: 303 COLUMN 65.
303: RIGHT PARENTHESIS FOR FUNCTION EXPECTED.
307: UNRECOGNIZED.
308: TOO MANY RIGHT PARENTHESES. NO MATCHING LEFT PARENTHESIS.

112 MAKEFMT FROMDATA=PROJECTS KEY=NOT RESULT=THERE NAME=$ACTID;
NOTE: THE PROCEDURE FORMAT USED 0.26 SECONDS AND 704K.
225 RUN;
NOTE: SAS USED 704K MEMORY.
ERROR: ERRORS ON PAGES 1.

FIGURE 13. Sample Execution of MAKEFMT With Nonexistent Input Variables

%MACRO CKVRS(DATASET,VRS),
"%* THIS MACRO CHECKS TO DETERMINE IF THE INDICATED 
%* VARIABLES EXIST ... ;
%* IN THE INDICATED DATASET. ... .
%* THE RETURN CODE IS SET IN &GOODVRS AS FOLLOWS: 
%* 0=AT LEAST 1 VAR NOT THERE 1=ALL VARS ARE there 
%* GOODVRS SHOULD BE MADE LOCAL BY THE CALLING MACRO. *;

DATA _NULL_; 
IF _N_=0 THEN SET &DATASET; 
ARRAY _N1 _NUMERIC_; 
ARRAY _C1 _CHARACTER_; 
RETAIN &VRS; 
ARRAY _N2 _NUMERIC_; 
DO OVER _N1; _NUM1+1; END; 
DO OVER _N2; _NUM2+1; END; 
IF _NUM1 NE _NUM2 THEN CALL SYMPUT('GOODVRS', '0'); 
ELSE CALL SYMPUT('GOODVRS', '1'); 
STOP; 
RUN; 
%IF &GOODVRS=0 %THEN 
%PUT ERROR: AT LEAST 1 VARIABLE DOES NOT EXIST IN DATASET &DATASET; 
%MEND CKVRS;

FIGURE 14. The CKVRS MACRO

Such a parser is fairly complex and the details are beyond the scope of this paper. However, the approach can be described by a simple example:

%MACRO PRINT(D1, D2, ... , D100)/STMT;
%Call to parser to get DATA= and YARS= ;
PROC PRINT DATA = &DATA; 
VARS &VARS; 
%MEND PRINT;

Note that all the parameters are positional. They are used to "capture" all tokens delimited by blanks after the MACRO name and before the semicolon(). Thus, the call:

PRINT VARS = A B DATA=X;

results in the following values for the positional parameters:
/* MACRO MAKEFMT(FROMDATA=, KEY=, RESULT=, NAME=)/STHT,
  */
/* *------------------------------------------------------------------* *
  * THIS MACRO CREATES A FORMAT MODULE FROM A GIVEN SAS DATA SET. IF*
  * THE DATASET IS EMPTY THEN A "NULL" FORMAT IS CREATED.
  */
*------------------------------------------------------------------* RUN;
%LET _ENTRIES=0;
%LOCAL DSGOOD GOODNAME ERR1 ERR2 GOODVRS;
%CKNAME(DATASET,&FROMDATA)
%LET ERR1=&GOODNAME;
%CKNAME(VARIABLE,&RESULT)
%LET ERR2=&GOODNAME;
%CKNAME(VARIABLE,KEY)
%IF &ERR1=0 OR &ERR2=0 OR &GOODNAME=0 %THEN
   %PUT ERROR: FORMAT &NAME NOT CREATED;
   %GOTO SKIPIT;
%END;
%CHECKDS(&FROMDATA)
%IF &DSGOOD=0 %THEN
   %PUT ERROR: FORMAT &NAME NOT CREATED;
   %GOTO SKIPIT;
%END;
%CKVRS(&FROMDATA,KEY &RESULT)
%IF &GOODVRS=0 %THEN
   %PUT ERROR: FORMAT &NAME NOT CREATED;
   %GOTO SKIPIT;
%END;
DATA _NULL_
SET &FROMDATA END=LASTREC;
MVAR='_FMT' I ILEFTCPUTC_N_,4.))
CALL SYR-IPUT(HVAR,' " ,
ITRIH(&KEY) II ' • '=" , II
TRI~1(&RESULT)
II ' )
IF LASTREC THEN CALL SYMPUT('_ENTRIES' .PUTC_K_,4.);
RUN
PROC FORMAT;
  OPTIONS NOTES;
  VALUE &NAME
%DO I=1 %TO &_ENTRIES;
    &&_FMT&I
%END;
%GOTO SKIPIT;
%END MAKEFMT;
FIGURE 15. MAKEFMT With Call to CKVRS to Verify That Variables Exist in Input Data Set

1 OPTIONS IMPMAC NOSOURCE2 TLS=72 LS=72 NODATE NONUMBER NOOVP;
2 %INCLUDE MAKEFMTS;
2
52 %INCLUDE CHECKS;
52
76 %INCLUDE CKNAME;
76
107 %INCLUDE CKVRS;
107
131 DATA PROJECTS;
132 INPUT ACT_ID $ ACT_TITL $&40.;
133 CARDS;
NOTE: DATA SET WORK.PROJECTS HAS 8 OBSERVATIONS AND 2 VARIABLES.
NOTE: THE DATA STATEMENT USED 1.10 SECONDS AND 640K.
ERROR: AT LEAST 1 VARIABLE DOES NOT EXIST IN DATASET PROJECTS
ERROR: FORMAT $ACTID NOT CREATED
142 MAKEFMT FROMDATA=PROJECTS KEY=ACTID RESULT=ACT_TITL NAME=$ACTID;
273 RUN;
ERROR: ERRORS ON PAGES 1.
FIGURE 16. Sample Execution of Revised MAKEFMT With nonexistent Input Variables

987
FIGURE 17. Sample Valid Execution of Revised MAKEFMT

FIGURE 18. Sample Call of MAKEFMT With Invalid Parameters.

<table>
<thead>
<tr>
<th>Macro Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>VARS</td>
</tr>
<tr>
<td>D2</td>
<td>=</td>
</tr>
<tr>
<td>D3</td>
<td>A</td>
</tr>
<tr>
<td>D4</td>
<td>B</td>
</tr>
<tr>
<td>D5</td>
<td>DATA=X</td>
</tr>
</tbody>
</table>

with D6-D100 being null. By searching for equals (=) signs, a parser can determine that the parameters given are VARS and DATA. These can be checked to determine if they are proper. The values for each parameter, VARS or DATA, is the text after the equals sign up to the next parameter. Using this logic results in the following values for the DATA and VARS macro variables:

<table>
<thead>
<tr>
<th>Macro Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA</td>
<td>X</td>
</tr>
<tr>
<td>VARS</td>
<td>A B</td>
</tr>
</tbody>
</table>

With the inclusion of such a parser, either specific to the called MACRO or a generalized parser, statement style MACROS can be developed which do fairly complete editing of their call.

4. CONCLUSION

In order to develop statement style MACROS, MACRO programmers should make their MACROS as self editing as possible. This tutorial presented some basic techniques to do this. Modules which perform specific functions can be developed to perform this editing. These modules can be MACRO code (as in CKNAME) or a mixture of MACRO code and SAS code (as in CHECKDS). The use of special techniques (e.g., IF N=0 THEN ... ) and options (e.g., NODATE,FORB) should not be overlooked when developing such modules.

The ORI author can be contacted at:

ORI, Inc.
Software Applications and Training
Program Office
7910 Woodmont Ave., Suite 1405
Bethesda, MD. 20814
(301) 656-3276