

Tips For Easing Frequent Data Set Variable Modifications

Randy C. Finch, Tennessee Valley Authority, Muscle Shoals, AL

ABSTRACT

In a plant research or production environment, many SAS® data sets containing operational information are generated. Even more data sets are generated as daily operational data are combined for analysis and reporting. In this environment, however, there are typically many changes and additions made to the plant, requiring the deletion of old variables and the addition of new variables in the data sets. Once the SAS programs have been changed, the new data sets can be operated on without problems. However, problems arise when using the new programs with older data sets. For example, some procedures (e.g. PROC PRINT and PROC TIMEPLOT) will abort with an error message if any of the variables to be operated on do not exist in the data set. Therefore, not only is the programmer faced with having to modify all the procedure calls in the reporting programs, he must also make sure that the older data sets are up-to-date so that they can be used by the procedures. This paper shows how macros can be used to simplify the formatting, and thus the changing, of reporting programs. It also shows how a simple SCL entry containing methods can aid in updating older data sets on an as-needed basis.

MACROS AND FORMATTED PROGRAMS

I maintain a SAS data management system at a plant research facility. As such, changes in the plant, and thus changes to the SAS data sets containing the plant data, are the norm. Most of the reporting is accomplished with SAS programs containing many procedure calls like the one shown here:

```
PROC TIMEPLOT MAXDEC=0;
  PLOT AFG_IVEN='I'
       AFG_RO='O'
       SFG_IVEN='S' /OVERLAY
  AXIS=0 TO 50000 BY 833.33333333
  REF=12500 25000 37500;
  ID STATUS;
  CLASS DTIME;
  LABEL DTIME="DATE:TIME"
        STATUS="STATUS"
        AFG_IVEN="AFG_IVEN(I)"
        AFG_RO="AFG_RO(O)"
        SFG_IVEN="SFG_IVEN(S)";
RUN;
```

When variable names are changed or variables are added or removed from the data sets due to changes in the research facility, many of these procedure calls have to be updated to match the new data. This is necessary because many SAS procedure calls such as PROC TIMEPLOT and PROC PRINT will generate an error if they are asked to use a variable that does not exist in the data set.

In order to simplify the process of changing the procedure calls, I decided to create macros to handle the procedure calls and then just define the necessary macro variables used by the macros. Listing 1 shows an example of a macro named TPLLOT that calls the TIMEPLOT procedure. To use this macro, 13 macro variables must be defined

and then the macro invoked. Listing 1 shows three examples of this process. The 13 macro variables that must be defined are the maximum decimal digits to use for the plot variables, the axis low, high, and step values, the reference line values, the variables to be plotted (up to four allowed), and the plotting symbols to use for each variable. After defining the 13 macro variables, the macro is invoked with the %TPLLOT statement.

Once the TPLLOT macro is invoked, it defines several macro variables used by the procedure. It checks to see which of the four plot variables have been defined and then assigns some text strings to be used in the TIMEPLOT procedure accordingly. Also, if more than one variable is to be plotted, the OVV macro variable is assigned a value of OVERLAY. This is used by PROC TIMEPLOT to specify that the plot variables should be plotted on the same graph.

By using the TPLLOT macro, all of the information needed by PROC TIMEPLOT can be laid out in a structured way by defining macro variables. All of the macro variable definitions are left-aligned after the equal sign and the ending semicolon is positioned at the same location (except for variables and symbols that are not being used) regardless of the actual length of the definition. This makes it easy to scan through the program and determine which variables are being plotted together. Also, it is easy to add new variables or move existing variable names and symbols to another plot should this be necessary. An editor or word processor (I use WordPerfect® for Windows™) that allows a rectangular block of text to be selected can be used to select variables and symbols as shown in Figure 1. The block can then be cut and pasted into another location without the need to reformat the text of the program.

UPDATING DATA SETS WITH METHODS

After the SAS programs have been modified to create and handle the new variables in the data sets, what is to be done about the older data sets that do not contain the newer variables? They cannot be used as is because some of the reporting procedures will abort with an error when one of the newer variables is referenced. In some cases, a separate set of SAS programs will have to be maintained to handle these older data sets. In other cases, as in mine, the older data sets can be updated to include the newer variables, even if their values are zero. The problem is when there are hundreds of older data sets needing updating. This is the situation I found myself in. There were many, many old data sets, and it was unknown which ones would need to be accessed again. Therefore, I had to update everyone of them each time changes were made to the data set variables. This was very time consuming and, in the case of the data sets that would never be used again, a waste of time. A way was needed such that when the user selected a data set to work on, it would be checked to determine if it was up-to-date or not, and then automatically update the file if necessary. This is where an SCL catalog entry containing methods came to the rescue.

I created an SCL entry named DSCHKUP.SCL containing two methods, CHECKDS and UPDATEDS. The former method is

responsible for checking a data set to see if it is up-to-date or not; the latter method is responsible for actually updating the data set if necessary. Listing 2 shows how these methods are actually used by a PROGRAM or FRAME entry in its SCL source code. At any point within the program where the user makes a data set selection, the variable THELIB should be set equal to the library in which the data set resides, and the variable THEFILE should be set equal to the data set name. Then the routine shown in Listing 2 should be executed with a LINK DFILE command. This routine first calls the CHECKDS method within the DSCHKUP.SCL entry, passing four parameters: THELIB, THEFILE, RMSG, and RVAL. THELIB is the name of the library containing the data set, THEFILE is the data set name, and RMSG and RVAL are variables that allow the method to return a message and an informational value. Make sure the latter two variables are initialized in the INIT section of the program or an error will occur when this method call is executed. When the method finishes checking the designated data set, the system message is equated to the method's return message and displayed. The CHECKDS method returns a value of zero if the data set is up-to-date and a value of one if it is not. Thus, if RVAL is one, the UPDATEDS method is called to update the data set. Finally, the return message from this method is displayed.

Listing 3 shows the code for the DSCHKUP.SCL entry. The key to the CHECKDS method is the assignment of the variable VAR2CHK. This is the variable whose existence will be checked in the selected data set. Each time the data set variables are modified, the variable to be checked has to be updated to the last added variable. The method assigns default return values for RMSG and RVAL and checks for the existence of the specified data set. If the data set exists, the VARNUM function is used to check for the existence of the variable designated by VAR2CHK. VARNUM returns the variable number of the specified variable. If the variable is not in the data set, it returns a zero. If the variable exists, the data set is up-to-date, and the appropriate message is returned along with an RVAL of zero. If the variable does not exist, the data set needs updating, and the appropriate message is returned along with an RVAL of one.

If the data set needs to be updated, then the UPDATEDS method is called. This method submits some SAS code. A data step is used to copy the selected data set to a temporary work space, delete variables that are no longer needed, and rename variables. If any of the variables to be deleted or renamed do not exist in the data set, a warning is issued but the data step will continue without aborting. Next, another data step combines a pre-built data set template containing no observations with the temporary data set. (The template is an empty data set already containing all the new variables in the appropriate order. By placing it first in the SET statement, all of the variables in the selected data set will be reordered according to the template.) Also, an external SAS program (PERFCALC.SAS) is executed to create all the new calculated variables. The resulting data set overwrites the original data set, leaving an updated data set in its place. Finally, an appropriate return message is assigned. At this point, the specified data set will contain all of the variables that a newly generated data set would have. This allows the reporting programs to reference the data set without error.

CONCLUSIONS

I have found these tips to be very helpful in maintaining data sets and SAS code when variables are changed frequently. I hope you find them useful, also. I would like to see a flag added to the reporting procedures in SAS that allows non-existent variables to be ignored while the existent ones are processed. This would eliminate a significant amount of code changes when data set variables are changed.

SAS is a registered trademark or trademark of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.

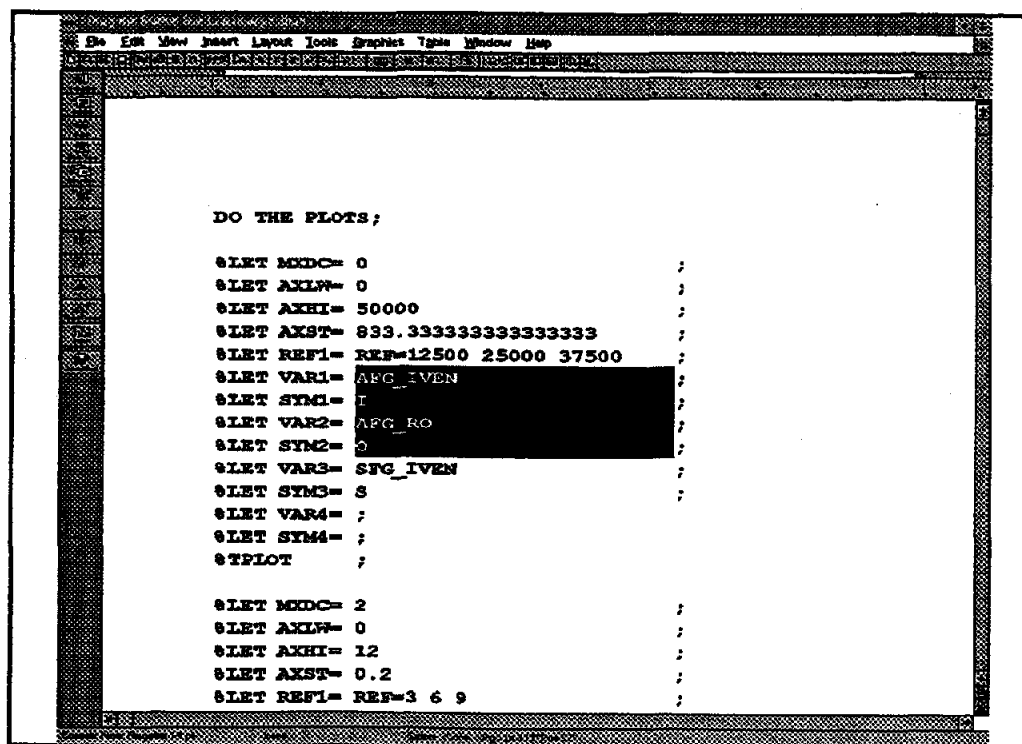


Figure 1. Rectangular Select in WordPerfect for Windows

Listing 1. Macro with Structured Statements

```

*****
*  EXAMPLE PROGRAM FOR SESUG '94 *
*  RANDY FINCH
*  TENNESSEE VALLEY AUTHORITY
*****;

* COPY THE DATA BASE;
DATA WORK.PLOTDATA;
  SET PRELIM.&SELFILE;
RUN;

* CHANGE THE DEFAULT PRINT OPTIONS;
OPTIONS LS=132 PS=58 NUMBER PAGENO=1;

* PLOT DATA MACRO;
%MACRO TPLOT;
  %LET EQ1= "&SYM1" ;
  %LET LB1= "&VAR1(&SYM1)" ;
  %LET OVY= ;

  %IF &VAR2= %THEN %DO;
    %LET EQ2= ;
    %LET LB2= ;
  %END;
  %ELSE %DO;
    %LET EQ2= "&SYM2" ;
    %LET LB2= "&VAR2(&SYM2)" ;
    %LET OVY= OVERLAY ;
  %END;

  %IF &VAR3= %THEN %DO;
    %LET EQ3= ;
    %LET LB3= ;
  %END;
  %ELSE %DO;
    %LET EQ3= "&SYM3" ;
    %LET LB3= "&VAR3(&SYM3)" ;
  %END;

  %IF &VAR4= %THEN %DO;
    %LET EQ4= ;
    %LET LB4= ;
  %END;
  %ELSE %DO;
    %LET EQ4= "&SYM4" ;
    %LET LB4= "&VAR4(&SYM4)" ;
  %END;

PROC TIMEPLOT MAXDEC=&MDC;
  PLOT &VAR1 &EQ1
        &VAR2 &EQ2
        &VAR3 &EQ3
        &VAR4 &EQ4
  / &OVY
  AXIS=&AXLW TO &AXHI BY &AXST &REF1 ;
  ID STATUS;
  CLASS DTIME;
  LABEL DTIME="DATE:TIME"
        STATUS="STATUS"
        &VAR1 &LB1
        &VAR2 &LB2
        &VAR3 &LB3
        &VAR4 &LB4 ;

RUN;
%MEND;

```

DO THE PLOTS;

```

%LET MDC= 0 ;
%LET AXLW= 0 ;
%LET AXHI= 50000 ;
%LET AXST= 833.33333333333333 ;
%LET REF1= REF=12500 25000 37500 ;
%LET VAR1= AFG_IVEN ;
%LET SYM1= I ;
%LET VAR2= AFG_RO ;
%LET SYM2= 0 ;
%LET VAR3= SFG_IVEN ;
%LET SYM3= S ;
%LET VAR4= ;
%LET SYM4= ;
%TPLOT ;

%LET MDC= 2 ;
%LET AXLW= 0 ;
%LET AXHI= 12 ;
%LET AXST= 0.2 ;
%LET REF1= REF=3 6 9 ;
%LET VAR1= RES_R ;
%LET SYM1= * ;
%LET VAR2= ;
%LET SYM2= ;
%LET VAR3= ;
%LET SYM3= ;
%LET VAR4= ;
%LET SYM4= ;
%TPLOT ;

%LET MDC= 0 ;
%LET AXLW= 100 ;
%LET AXHI= 400 ;
%LET AXST= 5 ;
%LET REF1= REF=175 250 325 ;
%LET VAR1= TD_RI ;
%LET SYM1= I ;
%LET VAR2= TD_RO ;
%LET SYM2= 0 ;
%LET VAR3= TD_EPI ;
%LET SYM3= S ;
%LET VAR4= TD_EPO ;
%LET SYM4= E ;
%TPLOT ;

```

Listing 2. Update Data Set -- Calling Code,
PROGRAM or FRAME

DFILE:

```

CALL METHOD('DSCHKUP.SCL','CHECKDS',
           THELIB,THEFILE,RMSG,RVAL);
  _MSG_=RMSG;
  REFRESH;
  IF RVAL=1 THEN
    CALL METHOD('DSCHKUP.SCL','UPDATEDS',
               THELIB,THEFILE,RMSG);
    _MSG_=RMSG;

RETURN;

```

Listing 3. Update Data Set -- DSCHKUP.SCL Entry

CHECKS:

```

METHOD SELLIB $ SELFIE $ RMSG $ RVAL 1;

* DEFINE THE VARIABLE TO CHECK FOR -- UPDATE AS NEEDED;
VAR2CHK='SR_A';

* DEFINE DEFAULT RETURN VALUE AND MESSAGE;
RVAL=0;
RMSG=' ';

* RETURN IF NO FILE SPECIFIED;
IF SELFIE=' ' THEN RETURN;

* OPEN FILE AND CHECK FOR LATEST VARIABLE;
SEL=SELLIB||'.'||SELFIE;
DSID=OPEN(SEL);
IF DSID=0 THEN DO;
  RMSG='Data set '||SEL||' does not exist.';
  RETURN;
END;
VARRTN=VARNUM(DSID,VAR2CHK);
RC=CLOSE(DSID);

* SEND CALLER APPROPRIATE MESSAGE;
IF VARRTN>0 THEN DO;
  RMSG='Data set '||SEL||' is current. No update required.';
  RETURN;
END;
ELSE DO;
  RMSG='Data set '||SEL||' needs to be updated. PLEASE WAIT!';
  RVAL=1;
  RETURN;
END;

ENDMETHOD;

```

UPDATES:

```

METHOD SELLIB $ SELFIE $ RMSG $;

SUBMIT CONTINUE;
  * CREATE TEMP FILE BY DROPPING AND RENAMING VARS IN OLD DATA SET;
  DATA WORK.TEMP;
    SET &SELLIB.&SELFIE;
    DROP   LBM_LFW
           LBM_LFAD
           PDIF_LF
           LBM_LFWL
          ;
    RENAME DUMMY02 = GPM_AMW
           DUMMY03 = GPM_ASFA
           DUMMY04 = GPM_AS
           DUMMY07 = TD_AMT
          ;
  RUN;

  * COMBINE MODEL FILE AND TEMP FILE AND EXECUTE PERF CALCS.
  THE MODEL FILE PROVIDES THE CORRECT VAR ORDERING AND BYTE SIZE;
  DATA &SELLIB.&SELFIE;
    SET UPMODEL.&SELLIB.WORK.TEMP;
    %INCLUDE 'PROGRAMS\PERFCALC.SAS';
  RUN;

ENDSUBMIT;
RMSG='Data set is now current. You may proceed.';
ENDMETHOD;

```