Building and Using %WINDOW Applications

Karen Crandall, Eastman Kodak Company, Rochester, NY

INTRODUCTION

SAS/AF® is a great tool for building complicated full-screen interfaces.For many applications, however, the user-interface is not particularly complicated and the macro window technology is a much better choice for implementing the window. Macro windows are easier to develop and debug. They are amazingly versatile.

This paper will briefly discuss the standard uses of macro windows and then move on to the more interesting problem of extracting a subset of data from a much larger dataset. The primary intent of this paper is to re-orient those who think that macro windows are only good for solving simple tasks by showing an example of how to do something that would have required a couple of extended tables if implemented in SAS/AF. The macro window solution is much easier.

BUILDING A MACRO WINDOW--THE ESSENTIALS

The easiest way to explore a macro window is to dissect one. Consider the example:

```
%WINDOW MENU
#1 @9 'Title'
#3 @5 opt1 1 a=underline +1
              'Option 1'
#4 @5 opt2 1 a=underline +1
              'Option 2'
#5 @5 opt3 1 a=underline +1
              'Option 3';

%DISPLAY MENU;
```

This example shows how to define a simple window and how to invoke it. The macro window definition begins with the keyword %WINDOW and the name of the window; it ends with a semicolon. Notice there is only ONE semicolon in a window definition. It does not matter how big or complex your window definition is, it has just one semicolon at the end.

When displayed, the window looks like this:

```
  Title

  _ Option 1
  _ Option 2
  _ Option 3
```

The user can select the options of interest by typing an X in the appropriate underscores. So that the program will have some way of knowing which options were selected, we assign each option a macro variable name (&opt1, &opt2, &opt3). Notice the syntax for doing this: specify the line and column to begin in, then the name of the macro variable, its length, and the underscore attribute. Next, move the pointer over a space, and specify some statement that defines what the option is. Repeat this for as many options as you have. Finish up with a semicolon.

Because the macro window syntax consists of just a few options, the code never gets too complicated. The next example shows how to use the window placement and color options. This is just about as messy as a macro window definition ever gets.

```
%window menu                rows=15 columns=52
                            irow=10 icolumn=14  color=blue

#1 @2    'Example of a Macro Window'                c=cyan
#3 @2    'Choose as many as you want'               c=cyan
#5 @2    makedata    1 c=white a=underline
                +1 'Generate test data'             c=yellow
#6 @5    prntdata    1 c=white a=underline
                +1 'Print test data'                c=yellow
#7 @5    plotdata    1 c=white a=underline
                +1 'Plot test data'                 c=yellow
#8 @5    re_plot     1 c=white a=underline
                +1 'Remove outliers & plot'         c=yellow
#9 @5    letmeout    1 c=white a=underline
                +1 'Quit'                           c=yellow;
```

The first thing you should notice about this example is how much it resembles the previous example. The default is for the window to occupy the entire display area. The intent of this example is to show how to center the window in the middle of the display. This code specifies the size of the window to be 15 rows and 52 columns and the window borders to be drawn in blue. The upper left corner of the window will be in coordinate (10,14).

There are two lines of text at the top of the window; the color on both is cyan. There are five options. The text of the options are in yellow, the underlines associated with each option are in white.

INTERFACING THE MACRO WINDOW WITH A PROGRAM

The program that interfaces with this window needs to initialize the window's macro variables, display the window, validate that the user has selected at least one option, and do some processing based on the user's request. If the user does not select something, this program also has to set up an error-condition and send a message telling the user what the problem is. Once corrected, the program has to clear the error condition and error message. Following is a program that shows how to accomplish these objectives.

```
%MACRO RUNJOB (dsn=testdata);

%* Initialize all MENU options to missing ;
%let makedata = ;
%let prntdata = ;
%let plotdata = ;
%let re_plot =  ;
%let letmeout = ;

%* Give user some instructions ;
%let sysmsg =
     Make choices, then press <ENTER>;

%* Display the menu and ... ;
%* make sure user selects something;
%let done = N;
%do until (&done=Y);
   %display MENU;
   %let done = Y;
   %if &makedata= and &prntdata= and
       &plotdata= and &re_plot= and
       &letmeout=
   %then %do;
         %let sysmsg =
              Make at least one choice;
         %let done = N;
         %end;
%end;

%* If user wants to print or plot test;
%* data then you have to generate some;
%* test data     ;
%if &makedata= and (&prntdata^= or
    &plotdata^= or &re_plot^=)
%then %let makedata = X;

%* Now we are ready to handle user ;
%* requests     ;
%if &makedata ^=
%then %do;
       DATA &dsn;
         RETAIN Y;
         %*deliberate outlier;
         X=1;  Y=1;  OUTPUT;
        Y=0;
        DO X = 50 TO 90;
          IF X < 71
          THEN Y = Y + 1;
          ELSE Y = Y - 1;
          OUTPUT;
        END;
        RUN;
       %end;
```

```
TITLE "Name of dataset being used
       is &dsn";
%if &prntdata ^=
%then %do;
      PROC PRINT DATA=&dsn;
        RUN;
      %end;

%if &plotdata ^=
%then %do;
      PROC PLOT DATA=&dsn;
        PLOT Y*X / VPOS=20
                   HPOS=60;
        RUN;
      %end;

%if &re_plot ^=
%then %do;
      PROC PLOT DATA=&dsn
           (WHERE=(x>20));
        PLOT Y*X / VPOS=20
                   HPOS=60;
        RUN;
      %end;

%if &letmeout ^=
%then %do;
      ENDSAS;
      %end;
%MEND RUNJOB;

%RUNJOB
```

USING A MACRO WINDOW TO SUBSET A LARGE DATASET

For many applications it is desirable to analyze a subset of a large dataset. The dataset itself is continuously updated with new data. The user wants to look at different subsets all the time. Therefore, the program cannot be hard-coded with an "index" of what is there, nor can it be hard-coded with subsetting logic. This kind of information is known only at runtime. For some reason, full-screen interfaces do not automatically come to mind for pre-processing types of problems. Developers tend to think of ANALYSIS when they think of full-screen interfaces. Nonetheless, a full-screen interface is the perfect solution to this type of problem. The following example shows how to implement a macro window to perform subsetting of data.

```
%MACRO DRIVER (dsn=original);

%* Setup (window) ;
%* 1. Initialize macro variables;
%*    for screen    ;
%do i = 1 %to 58;
   %let item&i = ;
   %let opt&i  = ;
%end;

%* 2. Using analysis dataset, ;
%*    list the unique item values;
%*    (these will be listed and the;
%*    user can pick the ones he   ;
%*    wants)          ;
PROC SORT DATA=&dsn OUT=UNIQUE
          (KEEP=ITEM) NODUPKEY;
  BY ITEM;
  RUN;

%* 3. Reinitialize macro variables ;
%*    from missing to whatever ;
%*    values appear on the dataset.;
%*    If fewer on dataset than on ;
%*    screen, leave the extra ones ;
%*    on the screen missing (so  ;
%*    they wont show up);
DATA _NULL_;
  SET UNIQUE END=EOF;
  ITEMNBR =
    'ITEM'||LEFT(PUT(_N_,2.));
  OPT     =
    'OPT'||LEFT(PUT(_N_,2.));
  CALL SYMPUT(ITEMNBR,ITEM);
  CALL SYMPUT(OPT,'_');
  IF EOF
  THEN CALL SYMPUT ('nobs',
                    PUT(_N_,2.));
  RUN;

%* 4. Define/Display the selection ;
%*    screen      ;
%*    Construct the screen such ;
%*    that columns of macro ;
%*    variables will appear (rather ;
%*    than rows) ;
%WINDOW SHOWME
#1  @7 'List of Items on Your File'
       c=cyan
#3  @7 'Select as many as you want
       and press ENTER'
       c=cyan
```

```
#6  @5  opt1  1 c=white +1 "&item1"
#7  @5  opt2  1 c=white +1 "&item2"
#8  @5  opt3  1 c=white +1 "&item3"
#9  @5  opt4  1 c=white +1 "&item4"
#10 @5  opt5  1 c=white +1 "&item5"
#11 @5  opt6  1 c=white +1 "&item6"
#12 @5  opt7  1 c=white +1 "&item7"
#13 @5  opt8  1 c=white +1 "&item8"
#14 @5  opt9  1 c=white +1 "&item9"
#15 @5  opt10 1 c=white +1 "&item10"

#6  @16 opt11 1 c=white +1 "&item11"
#7  @16 opt12 1 c=white +1 "&item12"
#8  @16 opt13 1 c=white +1 "&item13"
#9  @16 opt14 1 c=white +1 "&item14"
#10 @16 opt15 1 c=white +1 "&item15"
#11 @16 opt16 1 c=white +1 "&item16"
#12 @16 opt17 1 c=white +1 "&item17"
#13 @16 opt18 1 c=white +1 "&item18"
#14 @16 opt19 1 c=white +1 "&item19"
#15 @16 opt20 1 c=white +1 "&item20"

#6  @27 opt21 1 c=white +1 "&item21"
#7  @27 opt22 1 c=white +1 "&item22"
#8  @27 opt23 1 c=white +1 "&item23"
#9  @27 opt24 1 c=white +1 "&item24"
#10 @27 opt25 1 c=white +1 "&item25"
#11 @27 opt26 1 c=white +1 "&item26"
#12 @27 opt27 1 c=white +1 "&item27"
#13 @27 opt28 1 c=white +1 "&item28"
#14 @27 opt29 1 c=white +1 "&item29"
#15 @27 opt30 1 c=white +1 "&item30"

#6  @38 opt31 1 c=white +1 "&item31"
#7  @38 opt32 1 c=white +1 "&item32"
#8  @38 opt33 1 c=white +1 "&item33"
#9  @38 opt34 1 c=white +1 "&item34"
#10 @38 opt35 1 c=white +1 "&item35"
#11 @38 opt36 1 c=white +1 "&item36"
#12 @38 opt37 1 c=white +1 "&item37"
#13 @38 opt38 1 c=white +1 "&item38"
#14 @38 opt39 1 c=white +1 "&item39"
#15 @38 opt40 1 c=white +1 "&item40"

#6  @49 opt41 1 c=white +1 "&item41"
#7  @49 opt42 1 c=white +1 "&item42"
#8  @49 opt43 1 c=white +1 "&item43"
#9  @49 opt44 1 c=white +1 "&item44"
#10 @49 opt45 1 c=white +1 "&item45"
#11 @49 opt46 1 c=white +1 "&item46"
#12 @49 opt47 1 c=white +1 "&item47"
#13 @49 opt48 1 c=white +1 "&item48"
#14 @49 opt49 1 c=white +1 "&item49"
#15 @49 opt50 1 c=white +1 "&item50"

#6  @60 opt51 1 c=white +1 "&item51"
#7  @60 opt52 1 c=white +1 "&item52"
#8  @60 opt53 1 c=white +1 "&item53"
#9  @60 opt54 1 c=white +1 "&item54"
#10 @60 opt55 1 c=white +1 "&item55"
#11 @60 opt56 1 c=white +1 "&item56"
#12 @60 opt57 1 c=white +1 "&item57"
#13 @60 opt58 1 c=white +1 "&item58" ;

%DISPLAY SHOWME;

%* 5. Make a dataset which lists;
%*    the ITEM values user ;
%*    selected.  These values ;
%*    will be extracted from the ;
%*    analysis dataset and used in;
%*    subsequent processing.      ;
```

```
DATA KEEPME;
  LENGTH ITEM $ 8;
  %do i = 1 %to 58;
     %if &&opt&i ^= _ and &&item&i ^=
     %then %do;
           ITEM = "&&item&i";
           OUTPUT;
           %end;
  %end;
  RUN;

%* 6. Extract a subset from the original ;
%*    file for subsequent processing.     ;
PROC SORT DATA=ORIGINAL;
  BY ITEM;
  RUN;
DATA ANALZ_ME;
  MERGE ORIGINAL KEEPME (IN=KEEPME);
  BY ITEM;
  IF KEEPME;
  RUN;
PROC PRINT DATA=ANALZ_ME;
  TITLE 'User selected me for subsequent '
        'processing';
  RUN;
%MEND DRIVER;

**** Generate some test data ****;
DATA ORIGINAL;
  LENGTH ITEM $ 8 A B C $ 1 D E 2.;
  INPUT ITEM $ A $ B $ C $ D E;
  CARDS;
ITEM21    1    5    2    2    4
ITEM28    2    2    2    3    5
ITEM28    2    2    1   12   14
ITEM42    2    2    1    4    7
ITEM7     2    2    1    8    9
ITEM52    4    4    1   20   10
ITEM12    2    2    1    1   13
ITEM13    2    2    1    7   11
ITEM1     1    5    1   28   21
ITEM17    2    2    1   26   36
ITEM28    2    2    1    2    1
ITEM14    1    5    1   11    8
ITEM28    1    5    1   36   38
ITEM42    1    5    3   13   19
ITEM28    1    5    2    6   10
ITEM28    1    5    3   32   40
ITEM28    1    5    3   18   12
ITEM17    1    5    3    5   15
ITEM7     4    8    3   22    6
ITEM17    4    8    3   24   28
ITEM28    4    8    3    6   18
ITEM28    4    8    3   10   22
ITEM52    4    8    2   16   30
ITEM52    4    4    1   19   15
;
RUN;

%DRIVER
```

The dataset shown in this example contains a variable, ITEM. ITEM will be used as the subsetting variable. The macro window can accommodate up to 58 unique values of the variable ITEM. Each ITEM value will be preceded by an underscore; the user can

"X-off" those values of ITEM that are to be subsetted from the dataset.

The first step is to initialize the macro variables &item1-&item58 and &opt1-&opt58 to missing. Then determine the unique values of ITEM on the analysis dataset. Assign those values to the macro variables &item1-&item58. If there are fewer than 58 unique values, leave the excess macro variables at their initialized value (missing). For each &item# that has a non-missing value, initialize the corresponding &opt# to an underscore. Next, define a macro window. The window, SHOWME, will display up to six columns of ITEM values. If there are enough unique values to fill only two columns, then only two columns will be displayed. The magic that "hides" the unused columns can be explained by the fact that the unused &item# and &opt# macro variables are initialized to missing. Missing values are "displayed" as blanks -- that is, the user cannot see them.

The last step is to build the subsetted dataset. This is done by saving the values of the selected ITEM(s) in a "KEEPME" dataset, and then merging KEEPME with the analysis dataset and asking to keep just those ITEM values that appeared in the KEEPME dataset. The result is a dataset containing just those observations that need to analyzed.

Many applications require a finer level of subsetting than this. Let's say this dataset also included the variable INVENTRY, and INVENTRY was also needed to subset the data. A second macro window could be displayed to list the INVENTRY levels associated with the selected ITEM(s). The process for developing this second macro window is exactly the same as the one just described. Although the program would be longer, it really would not be more complicated because no new algorithm is necessary for this added functionality.

SUMMARY

SAS macro windows are easy to develop. Although they are most commonly considered for menuing types of applications, they are really more versatile than they appear. Often they can be used instead of a more complicated SAS/AF solution. The macro window syntax is relatively simple, so macro windows are not necessarily the answer to all user-interface problems, but they are certainly worthy of consideration.

Contact Information:

Karen Crandall
Eastman Kodak Company
Kodak Park
Rochester, NY 14652-4608
Telephone: (716) 722-6512