

WRITING SAS ® CODE BEHIND THE SCENES: A CONVERSATIONAL MACRO EXAMPLE

Deb Cassidy, Trilogy Consulting Corporation

ABSTRACT

Conversational macros provide a way for a non-SAS user to "write" SAS code when only base SAS is available. This paper will give you some advantages/disadvantages of this method as well as how to write the code.

INTRODUCTION

The communications department of the company often sent mailings to selected customers. All too often, the mailing labels were needed immediately so the correct amount of materials could be ordered. Although creating the mailing labels using SAS was a relatively simple task, it did mean an analyst had to stop working on another project. The communications department staff not only did not know SAS, but some of them barely knew how to turn on a PC to use any computer software. Although SAS/AF® or SAS/ASSIST® were designed to create applications for such situations, these products weren't available at the company. The SAS macro course provided a solution which required only base SAS - conversational macros. In a conversational macro, the macro facility prompts you at the terminal and accepts your input. The input is stored in macro variables which are then used in writing the program.

The code was originally written in Version 5.18. After I submitted the abstract, I discovered a change in the display manager in Version 6.06 that may make conversational macros awkward to use in some cases. The use of windows (new in Version 6.06) solves that problem and makes it easier for the user but the macro code does get more complicated. The concept is the same in both cases. I'll show both ways and you can decide which, if either, is right for your situation.

First, I want to go over some of the things to consider before you even write one line of code. Then I will cover how to write the code.

ADVANTAGES/DISADVANTAGES

Conversational macros have advantages and disadvantages. The main advantage for me was the ability to let non-SAS users "write" their own SAS code when only base SAS was available. Conversational macros also mean the user doesn't need to know such things as variable names, whether variables are character or numeric, if character variables are upper or lower case, dataset names, and how to write the procedure. This can save time even for an experienced user. The user only needs minimal training.

Of course, there are disadvantages.

- Someone has to write the macro in the first place and know all the information just mentioned. This could take more time than it is worth.
- You probably won't be able to include all situations - either because you can't identify them in advance or it's not worth the effort for extremely complex or rarely used situations. I once had a request for all the customers who had purchased a certain product within the last year but had not purchased any of the related products within the last six months. These customers also had to be eligible for the discount rate which was determined by their purchase of another set of products. This request was complicated enough without trying to write it as part of a macro for someone else to use!
- Conversational macros are interactive. Some sites discourage interactive programs on mainframes or minis because of the system resources required. If you have a very complex program or very large datasets to process, you may want to explore other alternatives. This is especially true if it's not practical to have the terminal locked up during processing.
- Different terminals will have different displays. Code must be written so it will work properly on any terminal that will be used.
- Conversational macros aren't very "pretty." V6.06 windows help the appearance.
- The use of the SUBMIT and ENTER keys can be confusing. All "answers" must be entered on line 1 of the Program Editor window

and submitted. V6.06 windows do let you use the ENTER key instead.

QUESTIONS

How complex is the code even outside the macro? What parts of the code will change for each use?

- Complex code with only 1 line that changes may be easier to write as a conversational macro than simple code that permits a lot of changes to titles and labels.

Can you write questions clearly so the user will be able to give a valid and correct answer? How much "HELP" information will you need?

- "Enter State code" seems simple enough. You enter AL but do you get Alaska or Alabama? What about special codes used only in your dataset? Would everyone know that your company identifies customer type with 1 for Medical, 2 for Law, and 3 for Academic?

How often will the conversational macro be used?

- Several times a month or only a few times a year? Since you probably won't be able to write every possible choice, how often will you have to write new code - either as a modification to the macro or as a separate job?

How many different programs will the user need to run?

- The more programs needed, the more likely SAS/AF will be worth the cost.

If conversational macros are still an option for you, the following shows you how to write some of the code.

WRITING THE CODE

The basic code for the mailing label is:

```
OPTIONS LS=100 PS=75;
DATA CUST; SET CUST.INFO;
  IF STATE='MO';
  POSITION='PRESIDENT';

PROC SORT; BY ZIP;
PROC FORMS DATA=CUST WIDTH=33 LINES=7
  ACROSS=3;
  LINE 1 POSITION/REMOVE;
  LINE 2 NAME;
  LINE 3 ADDRESS;
  LINE 4 CITY STATE ZIP/PACK;
```

(You may need to modify the options depending on your label size and printer. V6.06 options are shown here.)

Most of the code stays the same each time. You only need to change the IF statement and the value for POSITION.

How does the conversational macro work?

When the macro is invoked, %PUT statements will place text in the log window. The user will read the text and respond by entering a value on line 1 of the program window and submitting it. The response will be stored in a macro variable for later use. This process will continue until the complete job is "written". The results will appear in the output window unless they have been directed elsewhere such as an output file for printing.

The "behind the scenes" code is shown here in all caps while macro code is lower case. The %PUT statements will show the line in upper and lower case as entered. % and & will identify macro statements and variables. Everything else will appear in the job.

If only STATE could change, the macro would read:

```
%macro mail;
  %put Enter a 2-character state code;
  %input state;
DATA CUST; SET CUST.INFO;
  IF STATE="%upcase(&state)";
  POSITION='PRESIDENT';

PROC SORT; BY ZIP;
PROC FORMS DATA=CUST WIDTH=33 LINES=7
  ACROSS=3;

  LINE 1 POSITION/REMOVE;
  LINE 2 NAME;
  LINE 3 ADDRESS;
  LINE 4 CITY STATE ZIP/PACK;
%mend mail;
```

If you enter MO when asked, the macro variable state will have the value MO. The &state will resolve to MO and you will have the line:

```
IF STATE="MO";
```

You need to use double quotes to have the macro variable be resolved. The %UPCASE function

is used so you can input mo or MO or Mo or mO. If your original data are also upper and lower case, you should use:

```
IF UPCASE(STATE)="%upcase(&state)";
```

The rest of the examples assume the data are already uppercase.

Now, what happens if you want to use:

```
IF STATE="MO" OR STATE="OH" OR
STATE="MN";
```

```
%do i=1 %to 3;
  %put Enter 2-character state code;
  %input state;
  %if &i=1 %then
    IF STATE="%upcase(&state)" ;
    /* semi-colon ends %if */
  %else
    OR STATE="%upcase(&state)";
    /* semi-colon ends %else*/
  %end; /* ends do i for state;
  ; /* end IF statement */
```

Notice I've also mixed the styles of comments to show that both work. Comments are extremely important. I did not have as many comments in the original code. When I started to write the paper, I could not remember why I had a semi-colon in a certain spot. Semi-colons can be tricky. You need semi-colons to end the macro statements just like you need them for regular SAS statements. The semi-colon after the %END is necessary to end the IF statement you are creating with the macro.

What happens if you want to vary the amount of states? How do you tell the system:

```
%do i=1 %to some unknown value
```

There are actually several ways to accomplish this. I opted for the extra statements to ask how many states the user wanted. Although this option was easier for me, I selected it because it forced the user to think through the request before entering it. It also works as a safety check. If they wanted 5 states but only entered 4, there will be an extra "Enter state" and (I hope), the user will realize a state was missed. To fix the problem, the user will need to run the macro again for that state. A blank entered by accident should also be noticed.

```
%put Enter number of states;
%input numstate;
%do i=1 %to &numstate;
  (Same as above code)
```

Another option would be to have the user submit a blank line after the last state and use a %DO %WHILE or %DO UNTIL statement. In the macro language, there is no equivalent to the iterative DO with a while/until clause. I avoided this option because I felt it was more complicated to make sure I did not get a STATE=" " by accident since such a statement would retrieve some customers in foreign countries.

Believe it or not, that's really almost all you need to know. You would simply repeat the same code for other options.

What happens if the user wants to select by zip code instead of state. You don't want any SAS statements for state to appear in the final code. One way to solve this and add some error checking is to use macro label statements.

```
%statesel:
  %put Do you want to select by
state?;
  %put Enter YES or NO.;
  %input states;

  %if %upcase(&states)=YES %then %do;
    (code for selecting state - can
be imbedded in this code or a separate
macro )
  %end;

%else if %upcase(&state) ^=YES and
%upcase(&states) ^=NO %then %do;
  %put ERROR: &states is an invalid
entry.;
  %put %str( ); /* blank line for
legibility */
  %put Enter any letter to
continue.;
  %input cont; /* Keeps screen from
scrolling until you have read the
message.*/
  %goto statesel;
%end; /* end invalid entry */

%else %if %upcase(&states)=NO %then
%do;
  %put No state restrictions
requested.;
  %put %str( ); /* blank line for
legibility */
```

```
%put Enter any letter to
continue.;


```

```
%zipsel:
    (code for selecting zips)
/* You could have omitted the NO
section and the user would immediately
jump to the zipsel section after
answering NO. */
```

V6.06 WINDOWS

What happens if a set of %PUT statements will need more than one screen? In V5.18 it was easy to create a macro that would split the screen so only 1 line appeared in the program editor and the rest of the screen could be devoted to the log. V6.06 dropped one command that was necessary. Changing window sizes in MVS is now rather awkward so I recommend taking advantage of the new %WINDOW.

The %PUT statements wrote everything to the log window. %WINDOW lets you create your own windows which will be available for the session. A simple window is created by:

```
%window welcome
    #3 @15
'Welcome to the mailing label program'
    #6 @5
'Enter an X by the options to subset
your data'
    #10 @5 statex 1 @7 'STATE'
        @15 typex 1 @17 'TYPE'
    #14 @1
'Use TAB to move between options'
    #15 @1
'Use ENTER to move to the next screen'
;
```

Everything in quotes will appear in your window when you display it. This is the same as the %PUT statements except the entire screen will be used, previous statements will not appear and you can create several macro variables in one step. STATEX and TYPEX are 1-character macro variables which are used like the macro variables from %INPUT.

To display a window use:
%display welcome;

There are several options for %WINDOW statements such as color, underline, or reverse

video. Some options do not work on all terminals - I couldn't use any options when I wrote this! Although it works without options, I strongly suggest you underline the spaces for the input, if possible.

Using %WINDOW will also require some additional code. You will need %LET statements to initialize macro variables to null if you want the ability to submit more than one job per session. Once a macro variable is assigned, its value stays until it is assigned again. With the %PUT / %INPUT method, you do not refer to any macro variable you did not reassign.

PUTTING IT ALL TOGETHER

You can write the code as one long macro or you can write several shorter macros which are called by another macro. You should use the latter method if you:

- have many options to process ,
- may add options in the future,
- can use some options for multiple programs, or
- use %WINDOW.

You'll be able to work with each macro separately which will help with debugging. The system won't compile any macros that you don't call. In the case of %WINDOW, you can create several sets of mailing lists in a session without having to recreate the windows each time. I originally wrote one long macro (never expecting it to end up as 25 pages of code including all the %PUT %STR() statements.) Having written the long version, I'd recommend doing several shorter macros.

In V6.06, I used:

```
%macro mail;
    %mailwin; /* Creates all windows */
    %mailsel; /* Displays the windows
and writes the code. Other
macros are called from
this macro */
%mend mail;
```

The first step(s) to invoke the macro is site specific depending on the default options set for SAS and options for accessing stored code. You will need to have the proper library allocated. SAS will need to be invoked with the following options: MACRO, MAUTOSOURCE, and

SASAUTOS=libref. You can have the user enter necessary statements, create a CLIST (SAS is actually invoked with a CLIST), or other methods available at your site. Macros must be stored in a member with the same name.

Once you are in SAS display manager, enter

%MAIL

on line 1 and submit the line. Your log screen or V6.06 windows will tell you what to do next - just follow the instructions you wrote.

By default, the mailing labels will appear in the OUTPUT window. How to print them on mailing labels will vary by site. I had to save the output to a file and use a batch job to print the file.

OTHER NOTES AND COMMENTS

Here is some other code which many users may want in V6.06 but may have trouble writing.

The following writes an IF statement with the IN operator.

```
%do i=1 %to &numstate;
  %if &i=1 %then
    IF STATE IN("&state" ;
  %else
    , "&state";
  %end;
);
/* Notice the comma must precede the
next value or else you will have a
"dangling" comma. The right
parenthesis is added at the end just
like the semi-colon to finish the
statement;
```

%WINDOW also lets the user enter several values at once instead of a line at a time. In this window, the user could indicate one or more regions.

```
+ REGION-----+
| Place an X by all regions you want |
|                                     |
|   Northeast   Southeast   Midwest |
|   Mountain    Pacific     Foreign |
|                                     |
+-----+
```

How do you write the %DO loop when you don't

know which region will be first or which will be last? Here is one method. Each choice of region has been assigned a different macro variable name matching the data values. These values are NE, SE, MW, MT, PC, and FO.

```
%macro asgnreg;
  /* %global enables the macro
  variables to be used outside this
  macro. Macro variables are only
  available within this macro if
  listed in %local */
  %global kmax reg1 reg2 reg3 reg4
  reg5 reg6;
  %local k;
  %let k=1;
  %if &NE = X %then %do;
    %let reg&k=NE ;
    %let k=%eval(&k+1);
  %end;

  %if &SE = X %then %do;
    %let reg&k=SE ;
    %let k=%eval(&k+1);
  %end;
  (repeat for remaining regions)

  %let kmax=%eval(&k-1);
```

/* If the region has been selected, a new macro variable of the form reg&k will be created. K will be increased by 1 for the next macro variable. KMAX will be the maximum number assigned.;

```
%mend asgnreg;
```

```
%if &regsel=X %then %do;
  %asgnreg;

%do j=1 %to &kmax; /* 1 %to 1 works*/
  %if &j=1 %then
    IF REGION="&&reg&j" ;
  %else
    OR REGION="&&reg&j";
  %end; /* ends do i for state;
        ; /* end IF statement */
%end; /* ends if &regsel=X */
```

Another useful macro for mailing labels will determine the number of observations in the final dataset. This macro is explained in the writing utility macros chapter of SAS Guide to Macro Processing.

Don't forget to include the option of repeating the process without going all the way back to %MAIL. You may also need to include a way to

name the output file. Another helpful option for the user would be a verification screen. Whenever a macro variable has been assigned a value, that value will appear if you use the macro variable in a %WINDOW or %PUT statement. Verification is much easier to do with %WINDOW although it can be accomplished with %PUT.

While you are writing the macro, you may want to use options that show you what the macro facility is doing. MLOGIC will show you the logic of the code (ex. whether an %IF condition is true or false). SYMBOLGEN will show how macro variables resolve. MPRINT will let you see the code that is actually written. These options do add a lot to the log so you don't want to leave them on.

If you use V5.18 or V6.06 with windows, the function key should be SUBMIT. If you use V6.06 and the %PUT method, the function key should be SUBMIT;TOP.

OTHER OPTIONS

What about other options such as SAS/AF or CLISTS? A big advantage for both of these is the ability to create batch jobs which may be more desirable. SAS/AF can offer help screens along the way - for you to write the code as well as for the user. There is more flexibility in screen design. What's the catch - it also costs more! If you only have a couple jobs that users can run, then conversational macros will work. If there are many jobs, then drop by the SAS demo area to get more details.

The main disadvantage to CLIST is the limited flexibility. In general, the CLIST edits a job by issuing a series of FIND/CHANGE commands based on the information you entered. The edited version is submitted but the actual member is not changed so you can reuse it. CLISTS are useful if the code never varies but the values do. For example, I had a set of related reports where the only thing that changed was the time period in the title and in the dataset names. Depending on the month, I might run one or as many as seven jobs. The jobs also required 2 or 3 tape mounts which prevented interactive jobs. With the CLIST, I could submit the appropriate jobs as batch jobs without editing each one. The "menu" also reminded me which ones were needed that month. I could

also change the destination of the output because I could change the JCL.

CONCLUSION

Conversational macros offer one way of letting non-SAS users "write" code. There are advantages and disadvantages. This method is not useful for everything situation, but now you know what is involved and how to write the code.

A copy of the complete code is available by contacting the author.

REFERENCES

SAS Institute Inc. (1990), SAS ® Language: Reference, Version 6, First Edition Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1990), SAS ® Guide to Macro Processing, Version 6, Second Edition Cary, NC: SAS Institute Inc.

SAS, SAS/AF, and SAS/ASSIST are registered trademarks of SAS Institute Inc., Cary, NC

Deb Cassidy
3809 Secretariat Court
Columbus, OH 43221
(614) 771-7861 (messages)