Skipping, The Easy Way

Janet E. Stuelpner, ASG, Inc., Cary, North Carolina

ABSTRACT

The testing has begun. In order to see if the different parts of your SAS(®) program are working, you need to use PROC PRINT or some other procedure to output the data quite often. You practically have one after each DATA step or PROC step. What a pain in the neck it is to pull them all out when your testing is done and the program is perfect. Well, now you don't have to do it. A simple little macro will allow you to leave the source code there.

Another great use for this macro is to comment out many lines of code where documentation is plentiful. Sometimes it is difficult to remove sections of code with standard comments, where there are comments that exist . Read on to find out an easy way to skip over code with %skip.

INTRODUCTION

When I first learned about macros, the best way for me to use them was to put a %macro in front of a section of code and a %mend at the end. Usually, I would define only one parameter and then invoke the macro several times changing the parameter for each invocation. This worked very well. In one job, I could run off reports for many items, whether a particular system, class or investigator. I never thought that this basic premise would work so well in the future. With a slight modification, this action is extremely useful and a tremendous time saver.

The %skip macro is very simple and very useful. It does not need to be defined at the beginning of a job as we do with so many user written macros. It does not need any parameters. It can be defined as many times as you want within the same job. What makes this such a valuable tool is its versatility. There are many ways in which to use it. This paper will explore two of those ways

REASONS TO SKIP

Throughout the process of writing a program, it is necessary to view many of the intermediary data sets that are created in the process of running a job. The easiest way to do this is to print the output. When the program is completed, the printing is no longer necessary. Any procedure that creates output can be used. It is very difficult to do a search, because all of the output that are generated are not created in the same way. This is where %skip can be used. Define the macro and then invoke it. After testing is complete, remove the invocation. Because all of the output procedures have the same macro surrounding it, the search to remove the invocation is easy. The example below shows exactly how it can be done.

```
%macro skip;
PROC PRINT DATA=TESTA;
   TITLE1 'PRINT OF TESTA';
RUN;
%mend skip;
%skip;
        .
        .
        .
Other lines of code
        .
        .
        .
%macro skip;
PROC FREQ DATA=TESTB;
   TABLE MYVAR;
   TITLE1 'FREQUENCY OF MYVAR';
RUN;
%mend skip;
%skip;
```

When testing is complete, the %skip (which is the invocation of the macro) is removed. This can be done in two different ways: removal of the %skip or change it to %*skip. The whole macro can remain in place. You don't necessarily want to remove the whole thing. It may be necessary to test the program again at

some time in the future. This way the code for the output procedure can stay and can be used at any time. All you have to do is add the invocation back into the program.

WAYS OF SKIPPING

A. COMMENTING

There are two ways of writing comments to document a program. The first way is to begin with an asterisk and end with a semi-colon.

```
* comment ;
```

The other way begins with a slash asterisk and ends with an asterisk slash.

```
/* comment */
```

It is this second way that causes the most problems. Some programmers place a comment on each line of code to define what the code is doing. If there are a great deal of comments in the program and then it is necessary to comment out a large block of code, it can be very difficult to do. Since the comment ends with the first asterisk slash it sees, placing a slash asterisk before a block of code may end the code prematurely.

```
DATA AGE;
   SET DEMOG;
   AGEYR=(TODAY()-DOB)/365.25;
   /*CALCULATE AGE IN YEARS*/
   AGEGRP=PUT(AGEYR,AGEGP.);
   /*CATEGORIZE AGES*/
RUN;
```

Let's use the above code as an example. If it were necessary to comment out the whole DATA step, the first plan is to place a slash asterisk before the word DATA and then after the RUN. The problem with this is that the comment will end at the first asterisk slash (at the end of the word YEARS) and the next assignment statement will be flagged as an error (statement out of order). This is a good place for %skip. The use of the macro will avoid any error messages.

```
%macro skip;
DATA AGE;
   SET DEMOG;
   AGEYR=(TODAY()-DOB)/365.25;
   /*CALCULATE AGE IN YEARS*/
   AGEGRP=PUT(AGEYR,AGEGP.);
   /*CATEGORIZE AGES*/
RUN;
%mend skip;
```

If there is a need in the future to run this section of code, just invoke the macro. Using the macro like this avoids problems with ending a comment too soon and getting error messages.

B. PREVENT STEP FROM RUNNING

A RUN statement will execute any previously entered SAS statements. Typically it is used at the end of DATA steps and/or PROC steps. However, there is a little known option on the RUN statement which terminates the step without executing it. The system will print a message on the log indicating that the step did not run. An example of the use of this option:

```
PROC PRINT DATA=THEDATA;
   TITLE1 'JUST A PRINT';
RUN CANCEL;
```

CONCLUSION

So now you have seen two ways to use a very simple but powerful macro. An extensive background in macro development is not necessary. It is easy to use. It can be defined in the job stream. No parameters are necessary, but can be used if needed. Macros are extremely versatile. As can be seen above, it can be used to conditionally execute output statements or to comment out a block of code.

Janet Stuelpner
ASG, Inc.
326 Old Norwalk Road
New Canaan, CT 06840

(203) 966-7520 (voice)
(203) 966-8027 (fax)
jstuelpner@worldnet.att.net