Make Your Own Macro Processor

Ian Whitlock, Westat, Rockville, MD

Abstract

A simple macro variable processor which one might call SAS/PRETTYLOG is implemented in a short and simple SAS® program, PrettyLog.sas.

The example provides a better understanding of how SAS macro works while solving the problem of making a pretty SAS log, i.e. indentation preserved log with resolved macro variables and no SYMBOLGEN mess.

This utility only requires base SAS software and may be acquired at no more expense than a little learning effort.

Introduction

A colleague asked me how to design a C program to read and clean up a SAS log. It was essential to preserve indentation and make error messages come out at the relevant points and be able to see the resolved code in order to help debug a complex DATA step.

I thought about tackling the problem from the other end. Figure out how to modify the code so that the macro variables would be resolved before SAS processing. But preprocessing code is exactly what the macro facility is supposed to do. In short, I wanted my own little macro preprocessor because the options MPRINT and SYMBOLGEN do not produce a log meeting the above requirements, but the SAS log for pure SAS code does.

How could this be arranged? How about a SAS program to read an external code file and write the processed code to a catalog entry of type source? Then a simple %INCLUDE of that file would run the processed SAS program.

The only tricky part was how to resolve the macro variable references. Would it be cheating to use the SAS macro facility? As a teaching device about the macro facility, the answer is probably yes, but to simply solve the problem, why not?

The SAS function CALL EXECUTE can be used to execute the %LET statements as they are encountered during processing of the code, so that the macro variable references could be resolved in the processing step. Now the RESOLVE function can be used to resolve each line that contains ampersands.

Program

Here is the program.

```
filename inpcode
    "c:\nesug99\macPrettyTest.sas" ;
filename code catalog "work.macfac" ;

data _null_ ;
   infile inpcode truncover ;
   input line $char80. ;

   if left(line) =: '%let' then
      call execute ( line ) ;
   else
   if index ( line , '&' ) then
      line = resolve ( line ) ;

   file code (example2.source) ;
   put line $char80. ;
run ;

%inc code(example2.source) / source2;
```

Now we need a test program. It can be pretty simple since all we are testing is - can we read the program, locate the %LET statements, and resolve the macro references.

```
/* MacPrettyTest.sas */

   %let dir = c:\my documents\junk ;
   %let name = test ;
   %let xval = 7 ;

data _null_ ;
   file "&dir\&name..txt" ;
   x = &xval ;
   put "The number is " x ;
run ;
```

The astute reader can see that not any SAS program will work with this simple apparatus. For example, each %LET must be on a single line with nothing but spaces in front of it. Moreover, the resolved line must fit in 80 characters. In any case it is easy for many programs to meet these restrictions, when they are kept in mind. However, it is precisely these restrictions that make the problem of a pretty log simple. The problems with MPRINT come because the macro facility must be prepared to resolve macro variable references that can no longer fit on a line, since the resolved value can be up to 32K long.

The macro

The program could be packaged in a macro to simplify its use. Here it is.

```
%macro McPretty ( fileref=inpcode );

  options nomprint nonotes ;

 filename __cd catalog "work.macfac";

  data _null_ ;
     infile &fileref truncover ;
     input line $char80. ;

     if left(line) =: '%let' then
        call execute ( line ) ;
     else
     if index ( line , '&' ) then
        line = resolve ( line ) ;

     file __cd (example.source) ;
     put line $char80. ;
  run ;

  options notes ;
  %inc __cd(example.source)/source2;

%mend  McPretty ;
```

When the macro is in an autocall library the usage becomes:

```
filename mycode
    "c:\nesug99\macPrettyTest.sas" ;
%mcpretty(fileref=mycode)
```

Here is the log.

```
28   filename mycode "c:\NESug\macPrettyTest.sas" ;
29   %MacPretty (fileref=mycode)
NOTE: %INCLUDE (level 1) file __CD(example.source) is file
work.macfac.EXAMPLE2.SOURCE

30  +/* MacPrettyTest.sas */
31  +
32  +   %let dir = c:\my documents\junk ;
33  +   %let name = test ;
34  +   %let xval = 7 ;
35  +
36  +data _null_ ;
37  +   file "c:\my documents\junk\test.txt" ;
38  +   x = 7
39  +   put "The number is " x ;
        ---
        79
ERROR 79-322: Expecting a ;.

40  +run ;

NOTE: The SAS System stopped processing this step because of
errors.
NOTE: DATA statement used:
      real time           0.05 second
```

With the exception of wrapping to fit the columns one can see that our objective has been obtained. Note that the code submitted included a missing semi-colon to see how errors would be handled.

Conclusion

We started with a problem to obtain a pretty SAS log containing resolved macro variable references. The objective was met with an extremely simple SAS DATA step to preprocess SAS code with macro variable assignments and references in a manner very similar to that of the macro facility. This should alert one to the extreme simplicity of macro processing in general.

In part, we obtained the simplicity by using the macro facility. How essential was that use?

Two parallel arrays could be used to hold the macro variable names and their corresponding values to replace the macro facility symbol table. We could then for each line loop over the array of variable names looking for references and use the function TRANWRD to make the replacements. In this case a subroutine would be needed to load the arrays as %LET statements are encountered. One could even use other triggers such as "!LET" for commands and "#VAR" for macro variable references. In this case the code is more complex, but still not out of reach from a student trying to understand what macro processing means. In fact, I question whether a student should be learning macro if his SAS ability is not up to understanding the program outlined.

The technique presented is interesting both as a teaching tool and as a solution to obtaining a pretty log. However, one should recognize that it was possible because we accepted a restriction that the SAS macro facility cannot accept (that the resolved code of one line will fit on one line).

The author may be contacted by e-mail at

whitloi1@westat.com

or by mail at

Ian Whitlock
1650 Research Boulevard
Rockville, MD 20850

Questions are welcome.