A SAS* Code Formatter

William S. Calvert, Service Research Inc.
Malla R. Rao, National Institutes of Health

Abstract

Most high level languages like C, Pascal, have code formatters which format source code for ease of understanding and debugging. SAS*, however, does not have such a tool at this time. We have implemented a simple code formatter to format SAS code, which recognizes commonly used constructs. We have used SAS for convenience in transporting the code across platforms, with no operating system specific statements. Our formatter illustrates text parsing and could be easily extended or modified to accommodate other standards.

Introduction

There are currently numerous information systems that have been developed and implemented using the SAS System. They vary in size from a few modules to large complex systems using multiple routines[1]. Such systems require meticulous documentation and adherence to strict guldelines and rules in their development. This promotes ease of system maintenance and understanding. Most large systems use a common library of macros to create a meta language [1] which saves programmers development time and ensures uniformity of code. Since individuals have their own preference in the process of logic implementation, as well as in the manner and way they format their code, using a common formatting style establishes a uniformity in the source documentation. Individuals having their personal preferences in code development find it bothersome to adhere to code formatting rules and are more comfortable developing and debugging programs laid out in their own style. Documentation is essential for future system maintenance as well as for further development. It is well known that programmers hate to document. They would rather get started on their next project[3], than document the current project. Taking the cue from ideas that have been floated for automated documentation systems to relieve the programmer of the dull work of documenting [2], we have implemented a simple post processor code formatter through which we pass the final tested source code before archival.

It is routine practice to maintain a hard copy of all source code in conjunction with the project documentation. Both types of documentation are constantly cross referenced. If every programmer formats his code in his own style, it is difficult to follow complex code, especially with a high level of nesting and complex conditional statements. It is desirable to pass all source code through a code formatter, so that the entire systems' code adheres to the same layout standard. Since every organization establishes its own standard and style, we have implemented a very general purpose code formatter to illustrate the concept. The code formatter takes care of most commonly used constructs and can be easily modified and enhanced for other constructs that could become features in future SAS releases. Our purpose here is to illustrate the development of a text parsing routine which can be customized to other standards. We emphasize that this code formatter does not perform lexical analysis or grammar checking and assumes that the code to be formatted has been tested in its original form and is compile error free (though sometimes it is not).

We support a large number of researchers who use SAS primarily for management of data and data analysis. Many errors stem from improper matching of the if-then-else and the do-end blocks (especially when the levels of nesting get deep). It is a tedious task to debug such code in the free form we often see. We found the easiest way to find and fix these logic problems was to first properly indent the code. The code formatter automates this task.

How it works.

A formatter could be elegantly designed and implemented using languages like LISP, designed for list processing, or REXX or C. In our environment, we have SAS running on several different platforms. Since the formatter was to be used to format only SAS code, we decided to use SAS for its implementation.

The formatter makes two passes through the source code. The first pass separates multiple statements occurring on the same line. This process is complicated by commented lines, lines having quotes (dits), and those having link labels. See Table 1 and Listing 1. The output of this pass is a temporary working dataset which contains comments, lines with quotes, labels and SAS statements. This output dataset is the input for the second pass which performs the indentation of the code and writes out the final program file.

For persons wanting to follow and understand the code, Table 1 explains in a succinct manner, the way in which the flags are being set and reset in the first pass. Refer to the first data step in Listing 1 for the actual code. The main concept is that each line is separated at the occurrence of a semicolon or colon, unless they occur within a comment or quoted string, in which case the line is left 'as is'.

The second pass works under the assumption that the lines of code are logically separated. The formatting standard is applied by recognizing features of the code and placing them in the output file according to a predetermined pattern. Each line of code is indented two spaces (the value is stored in the indentation variable INCREMNT) when there is a change in logic level.

Job control language, "/*-*/" comments and "%*" comments are first written at position 1 if found. Comments of the form "* comment;" are identified and written starting at the current value of the pointer (the variable name is POINTER).

Link labels are identified as having a colon as the last non-blank character. The pointer is set to twice the indentation increment and the label is written starting at the pointer. In our example, (Listing 1) that makes link labels always 4 columns in from the left.

Next, the first word on the line is examined. If it is one of the 9 keywords (PROC, DATA, OPTIONS, CARDS, LABEL, MACRO, MEND, LET, and RUN) it is placed in position 1. Subsequent lines of parameters are indented by the 2 column increment.

Finally we place the DO/SELECT constructs. DO's can be the first or last word on a format line. DO's and END's are counted so the program can recover if the code being formatted has been damaged (or wasn't tested). Each level of logic is indicated by the 2 column increment of indentation.

Discussion

The process we present can be used even if you want to change the formatting standard from that which we have used. The value of INCREMNT can be changed if you want to modify the number of spaces the constructs are indented. The array of key words can be modified as well. The pattern we used is demonstrated in Listing 1. Listing 2 is the same code before going through the formatter.

Other keywords or constructs could be identified and structured in similar fashion. For example: BY, DROP, RETAIN, KEEP, SET, ARRAY, and others might beg for specific placement. Our program doesn't consider these since our primary need was for DO/SELECT, and IF/THEN/ELSE constructs to be formatted clearly.

We have produced a SAS program which converts unformatted SAS code to a more structured format. Not all constructs are planned for; however, this formatter features proper indentation of DO/END, and IF/THEN/ELSE constructs sensitivity to quoted strings, comments, and link labels. It may be used in any SAS environment, though we have noticed some differences across platforms. This basic formatter can be extended and/or modified to serve many formatting needs. Its usefulness is limited only by the time you have to invest in adding other features.

Bill Calvert
Service Research Inc.
401 Hull Place
Rockville, MD 20852

FAX (301) 738-1041

References

1. The Design and Programming of Large Systems using SAS Software, John M. Koman, 587-589, SUGI 12 Proceedings.

2. Automating Documentation: On How to Avoid the Really Dull Work and Still Know What You are Doing, 593-597, SUGI 12 Proceedings.

3. The Importance of Documentation, Stanley Voynar, 842-846, SUGI 14 Proceedings.

4. SAS Language: Reference, Version 6, SAS Institute Inc., Cary, North Carolina.

Table 1.

The approach used for scanning for dits, semis and colons. Tabulation of possibilities and actions:

p	sp	df	udf	cut	start	done
0	0	-	-	0	-	1
0	>0	0	0	sp+cp	-	1
0	>0	1	1	0	-	1
>0	0	0	1	I	cp+1	0
>0	0	1	0	0	-	1
	when 0<sp<p					
>0	>0	0	0	sp+cp	-	1
>0	>0	1	0	I	cp+1	0
	when 0<p<sp					
>0	>0	0	1	I	cp+1	0
>0	>0	1	0	I	cp+1	0

Where p is the position of the dit, sp is the position of the semi or colon and df is the dit-flag, udf is the updated dit-flag. CUT is the position at which the line is to be divided. Start is actually the cumulative position and the beginning of the interval to be scanned. DONE indicates when the calculation of the cut point has been accomplished.

Listing 1.

```
filename formfile 'd:\listing.one'/*LOCATION OF OUTPUT FILE*/;
%LET UPPER=0  /* IF ONE -  FORCE ALL OUTPUT TO UPPERCASE */;
  * 'increment' is the variable which holds the indentation interval;
options nocenter nodate  ;
  ;
DATA fcode1 ;
  infile 'd:\listing.two'/* IDENTIFY CODE TO FORMAT HERE */;
  /*
This code  formatter is intended  to  be used to put ordinary SAS
code in a regular format.  Specific features include:
    - Identification of DATA PROC OPTIONS LABEL AND CARDS statements
    - Proper indentation of DO/END and IF/THEN/ELSE constructs
    - Sensitivity to quoted strings using dits (')
    - When in doubt leave it alone orientation to JCL and other lines
    - Identification of LINK labels and SELECT constructs
    - Recognition of comments (of the form * comment text ; )
    - Recognition of large commented out blocks of code
    - May be used in any SAS environment, though we have noticed
      some differences across platforms
*/
INPUT  @1 C_LINE  $CHAR80. ;
label sp='position of semi-colon' cnt= 'position to divide line';
  retain passflag 0 ditflag 0 ;
  keep f_line passflag;
  * special handling of JCL lines if required ;
  if UPCASE(substr(c_line,1,2))= 'XX' then do ;
    substr(c_line,1,2)='//';
    f_line=c_line ;
    output fcode1 ;
    return ;
    end;
   /* routine added to deal with comment lines 9may91 mrr/wsc
      --- revised 23sep91 mrr/mkc/wsc */
;
if length(left(c_line)) ge 2 then do;
  *note sas error - left trims;
  if (substr(left(c_line),1,2) = '/*'or
  substr(left(c_line),1,2) = '%*') then passflag+1;
  if passflag  then do ;
    f_line=c_line ;
    output fcode1 ;
    if (substr(left(reverse(c_line)),1,2) = '/*' or
    substr(left(reverse(c_line)),1,2) = '%*') then passflag+(-1) ;

    return ;
    end ;
  end;
L=length(trim(c_line)) ;
do while(L gt 0);
  done=0 ;
  cp=0 ;
  do until(done=1) ;
    p=indexc(substr(trim(c_line),(cp+1),L-(cp)),'''');
    sp=indexc(substr(trim(c_line),(cp+1),L-(cp)),';:');
    if p=0 then do ;
      if sp=0 then cut=0 ;
      if sp gt 0 then do;
        if ditflag=0 then cut=sp+cp ;
        else if ditflag=1 then cut=0 ;
        end ;
       done=1 ;
       end ;
     ;
     if p gt 0 then do ;
       if sp=0 then do ;
         if ditflag=0 then do ;
           done=0 ;
           ditflag=1 ;
           end ;
         else if ditflag=1 then do ;
           cut=0 ;
           ditflag=0  /* RESET DIT FLAG HERE */  ;
           done=1 ;
           end ;
         end ;
       if 0<sp<p then do ;
         * do  problem corrected 17mar88  wsc ;
         if ditflag=0 then do;
           cut=sp+cp ;
           done=1 ;
           end;
         else if  ditflag=1 then do ;
           ditflag=0;
           done=0 ;
           end ;
         end ;
       if 0<p<sp then do ;
         if ditflag=0 then ditflag=1 ;
         else if ditflag=1 then ditflag=0 ;
         done=0 ;
         end;
       end ;
     cp+p;
     IF L-CP=0 THEN DO;
       CUT=CP ;
       DONE=1 ;
       END ;
     end /* OF UNTIL */  ;
   if (L-cut)*cut=0 then do ;
     f_line=c_line ;
     output fcode1 ;
     return ;
     end ;
   else if cut gt 0 then do ;
     f_line=substr(c_line,1,cut);
     c_line=substr(c_line,(cut+1),L-cut) ;
     output fcode1 ;
     L=length(trim(c_line)) ;
     end ;
   end /* OF WHILE */  ;
return ;
;
data _null_ ;
  file formfile noprint ;
  set fcode1 ;
  increment = 2
  ;
  /* change indent increment here */ retain pointer 1 endflag 0 ;
  * this screens for jcl and block comments ;
  if length(f_line) ge 2 then do ;
    if substr(f_line,1,2)= '//' or substr(f_line,1,2)= '/*'
    then  passflag =1 ;
    end ;
  if passflag=1 then do ;
    pointer=1 ;
    link printit ;
    return ;
    end;
  f_line=left(f_line);
  * screen out comments ;
  if substr(f_line,1,1) = '*' then do ;
    link printit ;
    return;
    end ;
  * screen for link labels;
  if length(trim(f_line))=indexc(f_line,':') then do;
```

```
   * make sure colon is last character ;
   pointer=increamt*2 ;
   link printit ;
   pointer + increamt;
   return ;
   end ;
  * scan for keywords ;
  A1='PROC' ;
  A2='DATA' ;
  A3='OPTIONS';
  A4='CARDS';
  A5='LABEL' ;
  a6='MACRO' ;
  a7='MEND' ;
  a8='LET';
  a9='RUN';
  ARRAY AC1(*) A1-A9;
  frstword=upcase(scan(f_line,1)) ;
  lastword=upcase(reverse(scan(reverse(f_line),1))) ;
  DO i=1 to 9;
    if frstword =ac1(i) then do ;
      pointer=1 ;
      link printit ;
      if ac1(i) ne a4 then pointer+increant ;
      return ;
      end ;
    end ;
  * scan line for do or select constructs;
  if lastword ='DO' or frstword='DO' or frstword='SELECT' then do ;
    endflag+1 ;
    link printit ;
    pointer+increant ;
    return ;
    end ;
  if frstword ='END' then do ;
    link printit ;
    pointer+(-increamt) ;
    endflag+(-1 );
    if endflag< 0 then do ;
      put @1 ' *** ERROR - More ENDs than DOs ... ; ' ;
      endflag=0 ;
      * setup to continue after error ;
      end ;
    return ;
    end ;
  link printit ;
  return ;
  ;

  Printit:
    IF &UPPER=1 THEN F_LINE=UPCASE(F_LINE) ;
    line_len=pointer + length(f_line);
    if line_len gt 72 then do ;
      put @2 f_line ;
      put @1
      ' *** ERROR formatted line greater than 72 bytes;' pointer=;
      end ;
    else put @pointer f_line $varying72. line_len ;
    return ;
run ;
```

Listing 2.

```
filename formfile 'd:\listing.one'/*LOCATION OF OUTPUT FILE*/;
%LET UPPER=0  /* IF ONE -  FORCE ALL OUTPUT TO UPPERCASE */;
* 'increamt' is the variable which holds the indentation interval;
          options nocenter nodate  ;;DATA fcode1 ;
  infile 'd:\listing.two'/* IDENTIFY CODE TO FORMAT HERE */;
  /*
This code  formatter is intended  to  be used to put ordinary SAS
code in a regular format.  Specific features include:
    - Identification of DATA PROC OPTIONS LABEL AND CARDS statements
    - Proper indentation of DO/END and IF/THEN/ELSE constructs
    - Sensitivity to quoted strings using dits (')
    - When in doubt leave it alone orientation to JCL and other lines
    - Identification of LINK labels and SELECT constructs
    - Recognition of comments (of the form * comment text ; )
    - Recognition of large commented out blocks of code
    - May be used in any SAS environment, though we have noticed
      some differences across platforms
*/
  INPUT  @1 C_LINE  $CHAR80. ;
label sp='position of semi-colon' cnt= 'position to divide line';
   retain passflag 0 ditflag 0 ;   keep f_line passflag;
   * special handling of JCL lines if required ;
   if UPCASE(substr(c_line,1,2))= 'XX' then do ;
     substr(c_line,1,2)='//'; f_line=c_line ;
     output fcode1 ;return ; end;

   /* routine added to deal with comment lines 9may91 arr/wsc
      --- revised 23sep91 arr/mkc/wsc */
  ; if length(left(c_line)) ge 2 then do; *note sas error - left trims;
  if (substr(left(c_line),1,2) = '/*'or
  substr(left(c_line),1,2) = '%*') then passflag+1;
  if passflag  then do ; f_line=c_line ; output fcode1 ;
  if (substr(left(reverse(c_line)),1,2) = '/*' or
  substr(left(reverse(c_line)),1,2) = '%*') then passflag+(-1) ;
 return ; end ;end; L=length(trim(c_line)) ;
do while(L gt 0);     done=0 ; cp=0 ;  do until(done=1) ;
 p=indexc(substr(trim(c_line),(cp+1),L-(cp)),'''');
 sp=indexc(substr(trim(c_line),(cp+1),L-(cp)),';:');
       if p=0 then do ;   if sp=0 then cut=0 ;
   if sp gt 0 then do;       if ditflag=0 then cut=sp+cp ;
     else if ditflag=1 then cut=0 ;     end ;   done=1 ;   end ;
 ; if p gt 0 then do ;   if sp=0 then do ;     if ditflag=0 then do ;
       done=0 ;  ditflag=1 ;  end ; else if ditflag=1 then do ;
       cnt=0 ; ditflag=0  /* RESET DIT FLAG HERE */  ; done=1 ;
end ;end ;if 0<sp<p then do ; * do  problem corrected 17mar88  wsc ;
     if ditflag=0 then do; cut=sp+cp ; done=1 ; end;
     else if  ditflag=1 then do ; ditflag=0; done=0 ; end ; end ;
     if 0<p<sp then do ; if ditflag=0 then ditflag=1 ;
     else if ditflag=1 then ditflag=0 ; done=0 ; end; end ; cp+p;
 IF L-CP=0 THEN DO; CUT=CP ; DONE=1 ; END ;
    end /* OF UNTIL */  ;
  if (L-cut)*cut=0 then do ;    f_line=c_line ;
    output fcode1 ; return ; end ; else if cut gt 0 then do ;
    f_line=substr(c_line,1,cut); c_line=substr(c_line,(cut+1),L-cut)
;
    output fcode1 ;    L=length(trim(c_line)) ;
    end ; end /* OF WHILE */  ;return ;   ;
 data _null_ ; file formfile noprint ; set fcode1 ; increamt = 2
   ; /* change indent increment here */ retain pointer 1 endflag 0 ;
   * this screens for jcl and block comments ;
   if length(f_line) ge 2 then do ;
    if substr(f_line,1,2)= '//' or substr(f_line,1,2)= '/*'
      then  passflag =1 ; end ;  if passflag=1 then do ;
     pointer=1 ;     link printit ;     return ; end;
   f_line=left(f_line);    * screen out comments ;
   if substr(f_line,1,1) = '*' then do ;link printit ; return; end ;
```

```
* screen for link labels;
 if length(trim(f_line))=index(f_line,':') then do;
     * make sure colon is last character ;
     pointer=increment*2 ; link printit ; pointer + increment;
     return ; end ;
   * scan for keywords ;   A1='PROC' ; A2='DATA' ; A3='OPTIONS';
 A4='CARDS'; A5='LABEL' ;   a6='MACRO' ; a7='MEND' ;
 a8='LET'; a9='RUN';
   ARRAY AC1(*) A1-A9;      frstword=upcase(scan(f_line,1)) ;
   lastword=upcase(reverse(scan(reverse(f_line),1))) ;
   DO i=1 to 9;     if frstword =ac1(i) then do ;  pointer=1 ;
 link printit ; if ac1(i) ne a4 then pointer+increment ;
 return ; end ; end ;
* scan line for do or select constructs;
   if lastword ='DO' or frstword='DO' or frstword='SELECT' then do ;
     endflag+1 ; link printit ; pointer+increment ; return ; end ;
   if frstword ='END' then do ;
     link printit ;     pointer+(-increment) ;
     endflag+(-1 );     if endflag< 0 then do ;
       put @1 ' *** ERROR - More ENDs than DOs ...  ; ' ;
       endflag=0 ;  * setup to continue after error ;
end ; return ; end ; link printit ;  return ; ;
   Printit:     IF &UPPER=1 THEN F_LINE=UPCASE(F_LINE) ;
     line_len=pointer + length(f_line);  if line_len gt 72 then do ;
       put @2 f_line ;   put @1
 ' *** ERROR formatted line greater than 72 bytes;' pointer=;
end ; else put @pointer f_line $varying72. line_len ; return ;
run ;
```