SAS® Comments – How Straightforward Are They?

Jacksen Lou, Merck & Co. , Blue Bell, PA

ABSTRACT

SAS comment statements typically use conventional symbols, '*', '%*', '/* … */'. Most programmers regard SAS commenting as a straightforward technique and are not fully aware of the difficulties that can arise from an improper use of comments. Occasionally, a misuse of comments can lead to program mistakes which are extremely difficult to find. This paper investigates how the SAS system processes comments and suggests a method to trouble-shoot some commonly seen commenting mistakes.

Key Words: SAS, Comments, *, %*, /* and */.

INTRODUCTION

SAS coders take advantage of and extensively use comments for a variety of reasons:

- Clarifying code for the reader
- Keeping original code
- Retaining undecided or controversial code
- Preserving proposed and experimental code
- And so on.

However, we frequently run into circumstances where a misuse of SAS comments leads to unfavorable outcomes, coding inconveniences, or bewildering mistakes, etc. Some of these problems are detectable and non-fatal, while others are fatal and extremely difficult to find.

Although the misuse of SAS comments has been repeatedly witnessed, no publication comprehensively covers the use of comments. Therefore, there is a need for a systematic investigation of the behavior of SAS comments, and for a method to trouble-shoot some commonly seen commenting mistakes.

The paper will discuss the following topics:

- Three typical commenting methods that use the special symbols of '*', '%*' and '/*…*/'
- Other non-conventional commenting techniques
- Trouble-shooting commonly seen commenting mistakes
- Good programming practices for using the commenting methods in a SAS macro

TYPICAL COMMENTING METHODS

Simply put, SAS commenting is text or code masking, i.e., making SAS code or text in a program inactive or non-executable. There are many different ways to achieve this. However, in routine programming practice, the following three typical SAS commenting methods are most frequently used.

1) SAS statement comment – starting with '*' at the beginning of a line of code/text and ending with a ';'.

2) SAS macro comment – starting with '%*' at the beginning of a line of code/text and ending with a ';'.

3) SAS block comment – starting with '/*' and ending with '*/'.

Statement Comment, <*…;>

This is a SAS statement comment, as distinguished from a SAS macro comment. The text commented in this way is treated, tokenized and processed by the SAS compiler or the processor as a complete but non-executable statement. Therefore, inserting a statement comment into a regular SAS statement will fracture its structural integrity. When such a fracture takes place an error or an unexpected outcome may occur because SAS cannot properly interpret either the inserted comment or the statement.

EXAMPLE

The integrity of this code was broken by inserting a SAS statement comment.

```
put *testing this; 'Patient ID = ' patno;
```

A SAS statement comment can be interrupted by inappropriately using a sensitive character such as a semicolon <;> or a single unmatched quote <’>.

EXAMPLE

This code does not work because it contains an internal semicolon.

```
*Statement comment is broken by a ';' here;
```

If a statement commented with ‘*’ is within a macro definition, the tokens of the statement will go through the macro facility before the SAS compiler recognizes it as a comment. The statement commented in this way is recognized as constant text and is stored in a compiled macro which is subsequently passed over to and is processed by SAS.

Like other constant text, the commented line within the macro definition will be printed to the log when the system options SOURCE and MPRINT are enabled. Therefore, within a macro definition, it is not desirable to use ‘*’ with text that is not intended to be seen in the log. Such a practice makes the log cleaner and neater for debugging and makes the processing more efficient.

If there are multiple ‘*’ on the same line, only the first ‘*’ is recognized as a special indicator of commenting. All the subsequent ones, regardless their positions in the comment, will be interpreted as a text character.

Under certain coding conditions, the text between **‘*’** and **‘;’** may not always be interpreted by SAS as commented text. For example, the following is a valid macro statement instead of a comment.

EXAMPLE

```
%put * I am by no means a comment;
```

From this example, we see that the comment ‘*…;’ is not standing alone, but related to a proceeding %put.

Macro Comment, <%*…;>

A macro comment can be used in a variety of ways.

Practice 1: in an open code macro statement, it is used to suppress macro processing.

```
%*put Today is &sysdate..;
```

Practice 2: in a macro call, it is used to mute a macro call.

```
%*SilentMacro(ds=);
```

Practice 3: within a macro definition, it is used to comment out the text.

```
%* This is boring header information.;
```

The SAS macro compiler recognizes but ignores text commented by ‘%*’. Unlike a SAS statement comment, a macro comment is not treated as constant text, thus, not tokenized and stored in the compiled macro. A macro comment is not passed to the SAS processor for execution and is not printed to the log. For that reason, unless you intentionally want to see commented text in the log, you should be acquainted with ‘%***’** as a legitimate macro commenting method. Disappointingly, we have observed that many macros don’t take advantage of this technique as frequently as they ought to.

To appreciate the different ways the macro compiler deals with both ‘%*’ and ‘*’**,** run the following code**.**

EXAMPLE

```
%macro diff;
  %*I am not existing in a compiled macro. Bye
bye.;
   *I am NOT a macro comment. See me in the log
file.;
%mend diff;
options mprint;
%diff;
```

Block Comment, ‘/*…*/’

Compared to ‘*’ and ‘%*’, a block comment ‘/*… */’ masks everything, with the highest degree of assurance. This is because it masks sensitive characters, such as ‘*’, ‘;’, ‘%’, unmatched “’” or ‘”’, etc. Block commenting can also enclose other comments, created using ‘*’ or ‘%*’ and make them regular text characters. This characteristic enables the block commenting to be used trouble-free anywhere in the code, even within a statement or a logic chain.

Unlike ‘%*’ and ‘*’ which are typically employed for a single line of code that ends with a semicolon, ‘/*…*/’ is handy for commenting out a large section of code with multiple lines of code, regardless of whether the code is open code or within a macro. Therefore, in macro testing or debugging, block commenting is more frequently used than any other commenting methods.

Nevertheless, a block comment cannot be nested within another block comment. For example, a nested format, </*… /*…*/ …*/>, simply does not work because it always stops at the first ‘*/’ while leaving the second ‘*/’ unresolved. This frequently causes some inconvenience to the programmer. Suppose a long program or macro used ‘/*…*/’ allover the code, it would be very demanding to test or debug because manually commenting out numerous sections of code that lie between pairs of </* … */> would be very wearisome and time consuming. On the other hand, using ‘%*’ or ‘*’ does not create this frustration, therefore, it would be more favorable to use them in the body of a long program or macro than to use ‘/*…*/’.

Block commenting is recognized and ignored by both the SAS compiler and the SAS macro compiler as a comment. The text enclosed by ‘/*…*/’ is not tokenized and is removed when the macro is compiled, thus the commented text does not show up in the log.

EXAMPLE

Submit the following code to see what happens in your log when SAS sees ‘*’, ‘%*’ or ‘/*…*/’.

```
%macro chckdata(dsin);

%if %length(&dsin) > 0 %then %do;
  %* infile given;
  data check;
  /* This dataset is used in proc means. */
    infile &dsin;
    input x y z;
    * check the data;
    if x<0 or y<0 or z<0 then list;
  run;
%end;
%else %put Error: No external data file is
specified;

%mend chckdata;

%chckdata(data1)
```

When you execute `%CHCKDATA`, the macro processor generates the following:

```
data check;
   infile ina;
   input x y z;
      * check data;
   if x<0 or y<0 or z<0 then list;
run;
```

This example shows that lines of text commented with ‘%*’ or ‘/*…*/’ are removed from the compiled macro, but the text commented with ‘*****’ still remains.

Caution: If you're operating in a batch environment on a mainframe, the machine may attempt to treat the ‘/*’ that starts in column one as JCL, while ‘*’ in column one is just fine.

OTHER COMMENTING TECHNIQUES

Key word ‘comment’

Probably in ancient code, you may see a commented line starting with the key word ‘**comment**’, instead of ‘*’ or ‘%*’. Key word ‘**comment**’ is still supported by SAS compiler, and is believed to be merely an early SAS syntax for ‘*’, and.

EXAMPLE

```
comment empty line;
```

Indeed, ‘**comment**’ acts exactly the same as ‘*’. However, this commenting technique is rarely seen in the coding practice now.

‘Virtual’ Commenting

There are also many other coding practices that prevent a certain section of code from being executed. These practices use regular SAS language and work effectively.

For example, you may play tricks to use a macro definition to comment out text or code which functions as documentation or reserved code.

EXAMPLE

```
%macro commenting;

*The following of code is reserved to compare
the current code;


     proc mixed data=_acttmp ;
          class &varx1 &varx2  ;
          model &_model &_modlopt  ;
          by &sumby  ;
         ... more code ...;

     run ; %* proc mixed  *;

%mend;
```

As long as the macro is not called all the code and text inside the macro will be treated as documentation, like the other commenting. The only difference is that the code will be compiled and the macro is stored.

Similarly, you may also use the following tactics to mask the code so that they are non-executable.

- subroutine GOTO
- conditional loop <IF… THEN DO; …; END;>
- macro loop <%IF… %THEN %DO; …; %END;>
- etc.

These techniques don't fall into the conventional notion of commenting. Therefore, we may call them 'virtual' commenting. Some people may find them handy and practical while others may think the tricks are too tricky because such techniques are too difficult to distinguish from true code, and are hard to read and to work with. In good programming practice, they should be avoided, especially in a finalized program.

SYSTEM OPTIONS AND COMMENTING

Some SAS system options can change the behavior of SAS commenting. The following are some of the most common ones.

```
<MPRINT>/<NOMPRINT>
<SOURCE>/<NOSOURCE>
```

It is well known that text commented with '*' may appear differently in the log when using the options of <MPRINT>/<NOMPRINT> or <SOURCE>/<NOSOURCE>. The system options 'MPRINT' and 'SOURCE' cause the text to be shown in the log, while 'NOMPRINT' and 'NOSOURCE' suppress the text from appearing in the log. It should be noted that the 'SOURCE' option is a generic option. It displays all SAS code, not just comments, in the log, while 'NOSOURCE' suppresses everything, including active code. Compared to <SOURCE>/<NOSOURCE>, on the other hand, <MPRINT>/<NOMPRINT> are only applied to the code handled by macro processor, thus more selective. The following code demonstrates the influence of these system options.

EXERCISE

```
options nosource; *source; /* please swap
between source and nosource */

data _null_;
x=1;
   /* commented out line */
y=2;
run;
```

TROUBLE-SHOOTING COMMON MISTAKES

1) A Broken Logic Chain

This is a mistake frequently made even by senior programmers. The error message resulting from this mistake is usually very hard to understand.

EXAMPLE

What happens if you run the following code?

```
%macro chgbase(checkvar);
   %if &checkvar=Y %then %do;
       %let _var=existing;
   %end;
   * This var has to be in the input data set.;
   %else %do;
       %let _var=no checkvar;
   %end;
%mend chgbase;
%chgbase(Y)
```

It generates the following errors.

```
ERROR: There is no matching %IF statement for
the %ELSE. A dummy macro will be compiled.
            %let _var=no checkvar;
        %end;
ERROR: There is no matching %DO statement for
the %END. This statement will be ignored.
        %put >>>&_var;
      %mend chgbase;

NOTE: SCL source line.
      %chgbase(Y)
      -
      180
WARNING: Apparent invocation of macro CHGBASE
not resolved.

ERROR 180-322: Statement is not valid or it is
used out of proper order.
```

Why did the errors occur? Here is the explanation.

As mentioned previously, SAS treats a '*' comment as an actual statement even though it is not executed. On the other hand, the code <%DO; …; %END; %ELSE; %DO; …; %END;> forms a complete logic chain, which does not allow a statement to be inserted between %END and %ELSE without breaking the macro logic chain.

In the above example, the statement, '* This variable …;', has effectively broken the chain of logic. When SAS sees %ELSE, it expects to locate a matching %IF (or %END). However, immediately prior to the %ELSE it encounters a statement comment, instead of %IF or %END.

To make the code work properly, the '*' statement comment should be replaced by the '%*' macro comment. This is because after the macro compiler removes the commented text, '%*This variable …;', during macro compilation, the logic chain of %END and %ELSE is kept intact. Similarly, '/*…*/' can also be used to achieve the same results in this circumstance.

As a matter of fact, any programming component that can be compiled, such as an extra semicolon, can also break the macro logic chain.

Such a problem can also occur in an open code logic chain, such as <IF…; ELSE;>.

EXAMPLE

The following 2 data steps do not work due to a break in the chain of logic.

```
data test;
  x=5; y=3;
  if x=1 then x=y**2;
  *recoding x;
  else x=y**2-1;
run;

data test;
  x=5; y=3;
  If x=1 then x=y**2;
  ;
  else x=y**2-1;
run;
```

2) An Interrupted Macro Definition

The definition of a key word macro is a continuous piece of code. This continuous piece of code can easily be interrupted by the misuse of a commenting method. The following examples illustrate some common problems.

EXAMPLE

Which of the commenting methods used in the code below has disrupted the following macro definition? Why?

```
%macro xxxx XXXX (DSIn=,      %* Input data set
;
             CutLngth=3, /* Var cut length */
             Debug=N     * Debug mode      *;
            );

  %put anything;

%mend xxxx;
%xxxx;
```

The comment preceded by '*' causes a problem because it has disrupted the definition even though it is placed at the end of the parameter list.

EXAMPLE

Which macro works and which does not?

```
%macro comp1 (arg1=
             ,arg2=
              %* The comment, with a comma;
             );
       %put arg1=&arg1;
       %put arg2=&arg2;
%mend;
%comp1();

%macro comp2 (arg1=
             ,arg2=
              /* The comment, with a comma*/
             );
       %put arg1=&arg1;
       %put arg2=&arg2;
%mend;
%comp2();
```

Macro %comp1 does not work because ',' is not completely masked, thus, has interrupted the completion of the macro definition.

Macro %comp2 does work because ',' was effectively masked by the block commenting method.

3) An Unmatched Syntax

The reserved SAS key words usually require the text that follows them to match a given syntax. Following a SAS key word, text commented by '/*…*/' or '%*' is recognized by SAS as a comment, but text commented by ‘*’ may not be recognized.

EXAMPLE

Which data step has a problem? Why?

```
data _put1_;
  put /*this is a test*/ 'test';
run;

data _put2_;
  put *this is a test; 'test';
run;

data _put3_;
  put %*this is a test; 'test';
run;
```

Data step _put2_ does not work because the key word PUT requires a name, a quoted string, an array name, etc., but not a ‘*’.

Data step _put1_ and _put3_ both work because text commented by ‘/*…*/’ and ‘%*’ are ignored by SAS compiler, and the key word PUT continues to search for the component that finishes the statement.

4) A Misuse of ‘%’*

Have you seen people comment code this way?

```
*%let a=2;
*%MyMacro(ds=demo);
```

Is this really a permissible SAS commenting method? It appears to be because your Enhanced Editor has turned the line preceded by ‘*%’ into a color that is designated to indicate commenting (by default, from blue to green). Such a ‘commenting’ method works in many cases whether or not SAS acknowledges it as commenting. This may explain why this ‘method’ is widely used.

Nevertheless, programmers should know that the syntax of a statement commented this way may not be complete. For example, `*%let a=2;` is not a complete statement because the semicolon, ‘`;`’ only ends the ‘`%let`’ but not the ‘*’. An extra semicolon is needed to end the ‘*’. A complete statement would be written as `*%let a=2;;`.

Sometimes missing the second semicolon can lead to an error or an unexpected outcome depending on the surrounding text because the statement may fallaciously go on to include the code following it.

Furthermore, even after you add one more semicolon it is still not a properly formatted comment within a macro definition. It is an executable statement! Unfortunately, in open code, it is interpreted as a SAS statement comment which is not different from a regularly formatted ‘*…;’ comment.

EXAMPLE

Before you test this code try to foresee what the results may be.

```
%macro comm;
   %put Who am I?;
  *%put Am I a complete comment line?;
%mend comm;
%comm;

*%put OK with open code;;
*put OK with open code;
```

EXERCISE AND A MACRO TEMPLATE

EXERCISE

The following code was designed for you to learn more about commenting techniques via hands-on practice. The code will tell you how different commenting methods actually behave and what it will generate. Based on what you may want to see, you may modify, delete certain parts of it, or add some of your own code during your trials.

```
/* Out-macro block comment */
* Out-macro comment;

%macro xx;
/* In-macro block Comment */
* In-macro Comment;
%* In-macro Macro Comment;

* %let exec = 1111 ;;
/* This statement is executed, if given an extra
semicolumn. If not, error. */
  %put &exec;

data ss;
  x='1';
  y=2;
  if y=2 then do;
     put '>>>' y;
  end;
  * y value;
  else do;
    put 'y is not equal to 2';
  end;
  call symput('x',compress(x));
run;

%if &x=1 %then %do;
    %let comment=X value is &x;
%end;
```

```
/* will break the chain if you use SAS statement
comment (*). Give it a try. */
%else %let comment=X value is not 1;

%put >>>&comment;

%mend;

options mprint mlogic symbolgen;
%xx;
```

A MACRO TEMPLATE

The following is a sample macro template that illustrates the selective use of different commenting methods.

```
%MACRO XXXX(DSIn=,      ❶/* Input data set      */
            CutLngth=,   /* Var length for cut */
            Debug=N      /* Debug mode          */
           );

❷%********************************************;
  %*    AUTHOR:                                 *;
  %*    DATE COMPLETED:                         *;
  %*    TITLE:                                  *;
   ********************************************;
❸ *    NARRATIVE:                              *;
   *     THE PURPOSE OF THIS MACRO IS TO ...    *;
   *                                            *;
   ********************************************;
  %*   EXPLANATION OF MACRO PARAMETERS:         *;
  %*                                            *;
  %********************************************;
  %*   SAMPLE CALL:                             *;
  %*                                            *;
   ********************************************;
   *   REVISION HISTORY                         *;
   *   DATE     :                               *;
   *   ANALYST  :                               *;
   *   CHANGES  :                               *;
   *                                            *;
   ********************************************;

❷%*----------------------------------------- *;
  %*   INITIALIZE THE LOCAL MACRO VARIABLES    *;
  %*-----------------------------------------*;
   %LOCAL _PGM;

  %*-----------------------------------------*;
  %*   A NOTE IN LOG INDICATING WHICH MACRO IS *;
  %*   BEING PROCESSED.                        *;
  %*-----------------------------------------*;
   %LET _PGM = PLN.STAT.SASMACRO(XXXX);
   %PUT NOTE: NOW EXECUTING &_PGM;

  %*-----------------------------------------*;
  %*   MORE CODE COMMENT                       *;
  %*-----------------------------------------*;

 %MEND XXXX;
```

❶ use '/*…*/' in the macro definition to elucidate macro parameters. This use will not break the continuity of the macro definition since the comments are not compiled. However, try to avoid using block commenting elsewhere in the macro because debugging can be bothersome due to the inconvenience encountered when you try to comment out the code you don't want to run.

❷ use '%*' in the sections that are not intended to be seen in the log when the macro is executed. This method should be used as much as possible within a macro.

❸ use '*' in the program header section that you may need to see in log. The use of this commenting method should be limited in a macro environment.

CONCLUSION

The most frequently used commenting methods are '*', '%*', and '/*…*/'. Each has pros and cons, and is more suitable than others under a given set of coding conditions.

Understanding their distinctive behavior during SAS compilation helps a user to make a wise selection among them, which reduces mistakes, and makes the code easier to read, test and debug.

It is usually safer to use '%*' and '/*…*/' than to use '*'. Commenting with '/*…*/' is always the safest.

Macro commenting with '%*' should be the first choice when commenting in a macro environment, except in the macro definition where block commenting '/*…*/' is preferred.

If it is hard to distinguish the active code from the commented code in the log, your abuse of comments deserves a comment.

ACKNOWLEDGEMENT

The author wishes to thank Richard Lowry for his critical review of the paper, thank Lei Zhang and Beilei Xu for encouraging me to write this paper and Xingshu Zhu for her initial comments and discussions regarding the contents of this publication.

REFERENCE

SAS Institute (1990), SAS Language, Cary, NC, USA, Reference Version 6, First Edition.

TRADEMARKS

CONTACT INFORMATION

Your comments are valued and encouraged. Please contact the author at:

Jacksen Lou
Merck & Co., Inc.
UNA-102
785 Jolly Road
Blue Bell, PA 19422
(484) 344-2236
jacksen_lou@merck.com