

Paper SC-006
Source Code Documentation: When Comments Get in the Way

Susan Myers, Ruben Chiflikyan, Mila Chiflikyan
RTI International, Research Triangle Park, NC

ABSTRACT

As programmers learn to write code, it is stressed that a well written program has comments describing every step of the process. In a perfect world an inherited program should be easy for one to understand what is being done and why. Many times historical information is included when a legacy system is used from one year to the next. In general this practice is invaluable; however when the comments grow larger than the code itself it can hinder one's ability to follow the flow of code. When the source code has extensive commenting, both descriptive and commented out unused code, it results in a log that is difficult read. The effort to manually remove comments can be tedious and time consuming. It also introduces the possibility of additional errors that exacerbate the debugging process.

We have developed a tool to automate the process of removing all comments from a SAS[®] program. Our goal is to produce a log file that is uncluttered with comments. In this paper we examine the challenges of identifying various methods of SAS commenting and show our approach to produce a comment free program. The original document remains intact but we also generate a clean comment free version that can be used in the debugging stage. We discuss some of the challenges in detecting comments buried in code, and how we approach it. We also demonstrate how this can aid the programmer by showing a side by side comparison of sas code; the original commented SAS statements against the resulting version created by our process with comments removed.

INTRODUCTION

Documentation standards in the computer programming industry call for extensive descriptive information regarding all parts of a program. This may entail header information with general overall purpose and input information, as well as more detailed comments throughout the source code.

"In general, you should include comments explaining what every subroutine does, what every variable does, and an explanation for every tricky expression or section of code." This is a guideline found at <http://ei.cs.vt.edu/~cs2604/Standards/Standards.html>.

Documentation can save a programmer much time if code was developed by someone else. Historical information may be included that explains important reasons for certain methods. It may also reveal reasons for doing something a specific way that may not be self-evident. While these approaches may aid persons that inherit programs for maintenance and updating, they can also get in the way of debugging problems.

There are times that a legacy program needs modifications. One common situation is that input data may change. Some programmers may remove discontinued variables by commenting them out rather than deleting them. Another common practice is copying old code, starting with two exact copies of the original code, commenting out one set while updating the second set. During the program development this may serve a very good purpose of helping the programmer remember how it was done originally. However, too many times programmers leave this "dead code" in place. After several such iterations and passing source code to new programmers, the result may be a very messy program that is difficult to follow.

The clean up process to remove unwanted comments may be laborious and even error-prone. If a set of comment delimiters is left in place it can create new program errors that complicate the debugging process. We have developed a program to strip away all comments in a SAS program leaving only active code. Not only is the source code easier to read, but the resulting log file is less cluttered with unnecessary comments.

THE APPROACH

We have included two approaches to the task of removing unwanted comments from a SAS program or log. Either method could be used to accomplish the same result. The first one relies on flags set when specific conditions are met. The second method also uses flags but is more modular in content.

LINEAR APPROACH

This program is intended to remove comments from SAS programs as well as from the SAS logs. It is based on manipulating flags, which are created for each type of specific situation. Changing the values of these flags allows effective identification and treatment of any specific situation.

The following flags are used in the program below:

open_c - comment is open or closed,
open_q - quote is open or closed,
open_dq - double quote is open or closed,
sm - semicolon is open or closed,
c - star-comment is open or closed, and
ch - data is present or not.

Below is the full text of the program with comments.

The text of the SAS program is read, and a temporary dataset is created. Each line of the SAS program is represented as one observation in the temporary dataset.

```
%let nmb=907;
filename in "C:\RTI\SAS\SESUG2009\test&nmb..sas";

data old;
length text $200;
infile in truncover;
input text $char200.;
run;

data temp_&nmb;
set old; /* all the flags should be retained from line to line since the comments could be spread across
many lines. Note that the program reads one text line at a time and sets all the flags to 0;

retain open_q 0 open_dq 0 open_c 0 sm 0 c 0 ch 0; * the program processes the text lines
one character in a time;

do i=1 to length(text);

    /* This part of the code is searching for comments of the "*" type */

    if substr(text,i,1) ^= '*' and substr(text,i,1) ^= ' ' then ch=1;

    /* Note that this statement checks for data availability starting from the first line of the processed
program. When data will be found the flag will be set. The statement is looking for "*" because the first
star in the processed code would indicate the beginning of the comment;

The statement below checks for the ';' in the text as the end of the SAS statement. The open_q,
open_dq and open_c should be equal to '0' to make sure that the ';' is not a part of the comment
or a literal. */

    if substr(text,i,1) = ';' and open_q=0 and open_dq=0 and open_c=0 then
    do;
        if c=1 then
        do;
            /* Note that If the ';' is found it would indicate an end of the star-comment if 'c' or
star-comment was identified. At this point, we reset the 'c' flag to 0 and set this ';' to blank*/

            c=0;
```

```

        substr(text,i,1)= ' ';
    end;
    sm=1; /* since the ';' is found we are setting the 'sm' flag to 1 */
end;

/* The following code looks for '*' as the beginning of the star-comment. It assures that
the cursor is not in the middle of the comment or a literal. 'sm' is set to 1 to indicate that it
is not in the middle of a mathematical formula, such as x=y*2. Note that the combination "*,
could be erroneously considered as the second type of the comment. */

if substr(text,i,1)= '*' and
(open_q=0 and open_dq=0 and open_c=0 and sm=1) then c=1;

/* Here the program looks for star-comments as the very first statement in the processed
program */

if substr(text,i,1)= '*' and ch=0 then c=1;

/* The following code looks for double quotes in the processed program and makes sure that
they are not a part of comment (open_c) */

if substr(text,i,1)= '"' and open_c=0 then
do;

/* The following logic checks whether the double-quote was not set earlier; if not then we set
open_dq to 1. But if the double-quote was encountered earlier then it would mean that we
have already found the closing double-quote and the open_dq has to reset to 0 */

    if open_dq=0 and open_q=0 then open_dq=1;
    else if open_dq=1 then open_dq=0;
end;

/* The following logic checks whether the quote was not set earlier, and if it is true, we set open_q to 1.
But if the quote was encountered earlier then it would mean that we found the closing quote and the
open_q is reset to 0 */

if substr(text,i,1)= "'" and open_c=0 then
do;
    if open_q=0 and open_dq=0 then open_q=1;
    else if open_q=1 then open_q=0;
end;

/* The following logic checks whether the beginning of the comment indicator is found and is not a
part of the literal. If this is case, then the comment open flag (open_c) is set to 1 */

if substr(text,i,2)="/*" and open_c=0 then
do;
    if open_q=0 and open_dq=0 then open_c=1;
end;

/* The following logic checks whether the end of the comment indicator was found and the beginning
of the comment flag is on. In this case, we clear the end of the comment indicator and set the flag
(open_c) to 0, indicating that the comment is closed */

if substr(text,i,2)= "*/" then
do;
    if open_c=1 then
    do;
        substr(text,i,2)= ' ';
        open_c=0;
    end;
end;

```

```

/* This part contains comment-removing logic. If comment or star-comment is open, and
the cursor is not in the middle of the quoted string, then the comment characters will be cleared
one digit in a time */

if (open_c=1 and open_q=0 and open_dq=0) or c=1 then
  substr(text,i,1)=' ';

/* This part of code checks if ';' was found and the new statement started. In this case the semicolon
flag should be set to 0. Note that the non-blank and non-semicolon character would indicate the
start of a new statement */

if sm=1 and substr(text,i,1) ^= ' ' and c ^=1 and open_c=0
and substr(text,i,1) ^= ';' then sm=0;
end;
run;

/* This data step re-creates the original SAS program without comments.
Note that the positions of all SAS statements are unchanged */
data _null_;
set temp_&nmb;
x=verify(text,' ');
file out;
put @x+1 text;
run;

```

MODULAR APPROACH

This program has the same functionality as the first one but is based on a modular approach. It has both advantages and disadvantages relative to the linear version. Modification to parts of the code without affecting the rest is much easier; however, we have more code repetition.

```

%let nmb=908;

/* Input file to clean */
filename in "Z:\master\fics\ruben\prg&nmb..sas";

/* Output file to with comments removed */
filename out "Z:\master\fics\ruben\prg&nmb._clean.sas";

```

Here the text of the program. Note that the flags used here are as follows:

flag_eq is for finding '=' sign,
flag is for identifying '/*' and '*/',
flag_s is for identifying '*' and ';',

```

%let lg= 200; *declare the length of each input line;
data temp(keep=text1 textn); * we keep both the original and cleaned version of program lines;
length text1 $&lg.; * defining the length of the new string;
infile in truncover; * reading of the SAS code as a text file;
input textn $char%eval(&lg+1).; * the whole line is treated as one character variable;
retain flag 0 flags 0 flg_eq 0 Icur Istartn; * these flags and variables will be retained;

Istartn=1; * initialization of starting index;

start:; * the label of the goto loop below;
Istart=Istartn; * assuming the current index as the start value ;

do I=Istart to &lg; * starting loop to go over the string;
  if substr(textn,I,1)='=' then flg_eq=1; * if '=' is encountered flg_eq is 1;

```

```

else if substr(textn,I,1)=';' then flg_eq=0; * otherwise it is zero;

/* This section applies when comments of the second type are present */

if substr(textn,I,2)='/*' or flag=1 then
do; Icur=I; /* flag is initialized if '/' is encountered for the
first time or the cursor is still within the comment
area opened at some moment earlier */
do I=Istart to &lg; * starting of the new loop;
if I>=Icur then
do; * defining the area of nontrivial changes;
if substr(textn,I,1)='/' and substr(textn,I+1,1)='*' then flag=1;
* condition when flag is assigned to 0;
else if substr(textn,I,1)='/' and substr(textn,I-1,1)='*' then
do; * condition when the comment is closed;
flag=0; Istartn=I+1; * redefining flag and the outer loop star value;
substr(text1,I,1)=' '; * assigning space;
goto start; *go to label start;
end;
if flag=1 then substr(text1,I,1)=' '; * assigning space;
end;
end;
end;

/* This section applies when double quotes are present */

else if substr(textn,I,1)='"' then
do; Icur=I; * double quote is found;
do I=Istart to &lg; * start of the new cycle;
if I>=Icur then
do;
substr(text1,I,1)=substr(textn,I,1); *every character from the
original string is copied to the new one;
if substr(textn,I,1)='"' and I ^=Icur then
do;
Istartn=I+1;
goto start; * return back in case the next double quote is found;
end;
end;
end;
end;

/* This section applies when single quotes are present */

else if substr(textn,I,1)=''' then
do; Icur=I; * single quote is found;
do I=Istart to &lg; * start of the new cycle;
if I>=Icur then
do;
substr(text1,I,1)=substr(textn,I,1); * every character from the
original string is copied to the new one;
if substr(textn,I,1)=''' and I ^=Icur then
do;
Istartn=I+1;
goto start; * returns back in case the next single quote is found;
end;
end;
end;
end;
end;

/* This section applies when comments of the second type are present */

```

```

else if (substr(textn,I,1)='*' and flg_eq=0)
or (flags=1) then
do; Icur=I; * identifying the beginning of the comment;
  do I=Istart to &lg; * looping through all the fields;
    if I>=Icur then
      do; * the beginning of the comment;
        if substr(textn,I,1)='*' then flags=1;
        if substr(textn,I,1)=';' then
          do; *the comment ends with a semicolon;
            flags=0;
            Istartn=I+1;
            goto start; * leaving the loop;
          end;
        if flags=1 then substr(text1,I,1)=' '; * characters
          are replaced with blanks if they are positioned within a comment;
      end;
    end;
  end;

/* This part copies all the characters in case “*..” is not a comment */

else if (substr(textn,I,1)='*' and flg_eq=1) then
do;
  substr(text1,I,1)= substr(textn,I,1);
end;

/* Copy the string as it is if the cursor is out of the areas mentioned above */

else if substr(textn,I,2) ^= '/*'
and substr(textn,I,1) ^= '"'
and substr(textn,I,1) ^= "'"
and substr(textn,I,1) ^= '*'
then substr(text1,I,1)= substr(textn,I,1);
end;
if text1='' then delete;
run;

/* This data step creates the cleaned version of the program without comments */

data _null_;
set temp;
x=verify(text1,' ');
file out;
put @x+1 text1;
run;

```

RESULTS

We now demonstrate several examples where the programs remove the comments from the input SAS programs. Note that the programs can be applied to SAS logs as well as SAS programs. The following examples were run with both program versions provided, linear and modular, with similar results. One difference between the two programs is the linear version retains blank lines, whereas the modular one removes them.

INPUT WITH '*' BUT NO COMMENTS

Suppose that initial text contains no comments but it has asterisks embedded in mathematical expressions. Say the text in the prg01.sas program is as follows:

```
Data prg01;
  x=2;  y=x*3;
  z=2*x+3*y;
run;
```

Note that the some lines have preceding spaces. After running the code, the following cleaned version of the program (prg01.sas) results:

```
Data prg01;
  x=2;  y=x*3;
  z=2*x+3*y;
run;
```

We see that the text of the program did not change. For this example the code does not alter the text if there are no comments in it. Note that the code identifies that the strings ‘*3;’ and ‘3*y;’ are constituent parts of mathematical expressions rather than the comments of the second type.

INPUT WITH ‘/*...*/’ COMMENTS

In this case, we consider a standard SAS program (prg02.sas) containing ‘/*...*/’ comments, which could be located on one line and/or spread over many lines.

In this specific case, the SAS program with comments has the following form:

```
/* This is comment
spread across
many lines */

Data prg02; /* comment after a statement */
/* comment before a statement */ x=2;
/* comment before a statement */ y=2; /* comment after
a statement a spread across
many lines*/
run;
/* Another comment after the program */
```

After using our program, the cleaned version is as follows:

```
Data prg02;

                                x=2;
                                y=2;

run;
```

Note that the program leaves the SAS code statements in their original locations (columns). If necessary, all indented lines can be left-justified with a simple update to the last data step that writes the comment-free file.

INPUT WITH ‘*...;’ COMMENTS

In this case we consider a standard SAS program (prg03.sas) containing ‘*...;’ comments, which could be located on one line and/or spread over many lines.

Say, the contaminated version has the following form:

```
* This is comment of the second type
spread across
many lines ;

Data prg03; * comment after a statement;
* comment before a statement; x=2;
* comment before a statement; y=2; * comment after
a statement a spread across

many lines;
run;
    This is another comment after the program ;
```

After applying our program, the cleaned version (prg03.sas) is as follows:

```
Data prg03;

                                x=2;
                                y=2;

run;
```

COMMENTS INSIDE THE VALUE OF THE CHARACTER VARIABLE

In this case we consider a standard SAS program (prg04.sas) containing both types of comments, as well as some comments that are part of character strings. The commented version has the following form:

```
Data prg04;
x="asd /*hgf*/ dfg";
x="asd *hgf; dfg";
/* asdgfvjn "gd jhf" */

y='asd * hgf ; dfg';
run;
```

After running our program, the cleaned version takes the following form:

```
Data prg04;
x="asd /*hgf*/ dfg";
x="asd *hgf; dfg";

y='asd * hgf ; dfg';
run;
```

CONCLUSION

We acknowledge the need for well documented source code, but have experienced the reality where extensive commenting impedes a programmer's task of understanding legacy code. Our programs to remove comments from a SAS program or log are intended as a tool for any level programmer. By providing two methods we demonstrate the versatility of base SAS, and hope they serve a basic but useful purpose.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Susan Myers
RTI International
3040 Cornwallis Road, PO Box 12194
Research Triangle Park, NC, 27709-2194
Work Phone: (919) 541-7441
Fax: (919) 990-8368
E-mail: smyers@rti.org
Web: <http://www.rti.org>

Ruben Chiflikyan
RTI International
3040 Cornwallis Road, PO Box 12194
Research Triangle Park, NC, 27709-2194
Work Phone: (919) 541-6064
Fax: (919) 990-8368
E-mail: rchiflikyan@rti.org
Web: <http://www.rti.org>

Mila Chiflikyan
RTI International
3040 Cornwallis Road, PO Box 12194
Research Triangle Park, NC, 27709-2194
Work Phone: (919) 316-3360
Fax: (919) 990-8368
E-mail: milachif@rti.org

Web: <http://www.rti.org>

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.
Other brand and product names are trademarks of their respective companies.