Paper 1556-2014

Making the Log a Forethought Rather Than an Afterthought

Emmy Pahmer, inVentiv Health Clinical

ABSTRACT

When we start programming, we simply hope that the log comes out with no errors or warnings. Yet once we have programmed for a while, especially in the area of pharmaceutical research, we realize that having a log with specific, useful information in it improves quality and accountability. We discuss clearing the log, sending the log to an output file, helpful information to put in the log, which messages are permissible, automated log checking, adding messages regarding data changes, whether or not we want to see source code, and a few other log-related ideas. Hopefully, the log will become something that we keep in mind from the moment we start programming.

INTRODUCTION

By default SAS® provides a useful log that tells us, most importantly, when there are errors or warnings, so it's easy to feel like you don't need to do anything special about it. It just takes care of itself. However, there are times when you might feel that there is too much information to wade through, not enough information, or not the right information. For industries that are highly regulated and accountable, we have to be sure that there is documentation to support everything we do. We should consider our logs to be part of that documentation. The log can also help us ensure the quality of our data by documenting what we do to it.

CLEARING

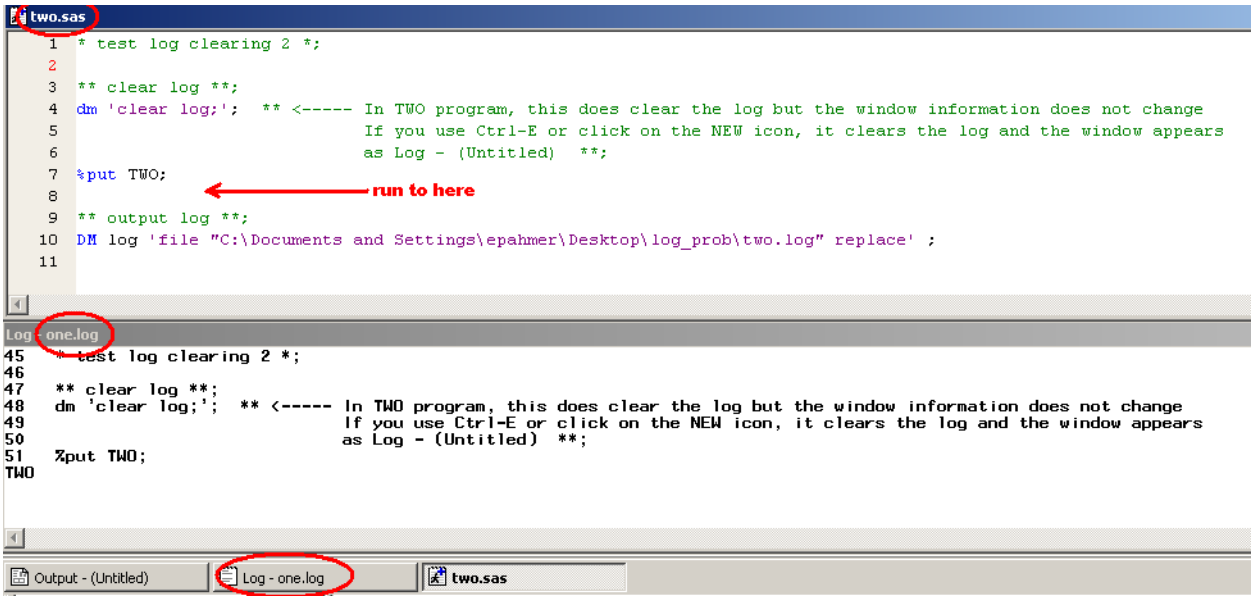
The log should be cleared at the beginning of the program to be sure that you don't end up with log information from a program that ran previously. I prefer to do this right at the top of the program, so that everything else is included in the log, including the program header.

This code clears both the log and the output window.

```
dm 'clear log; clear output';
```

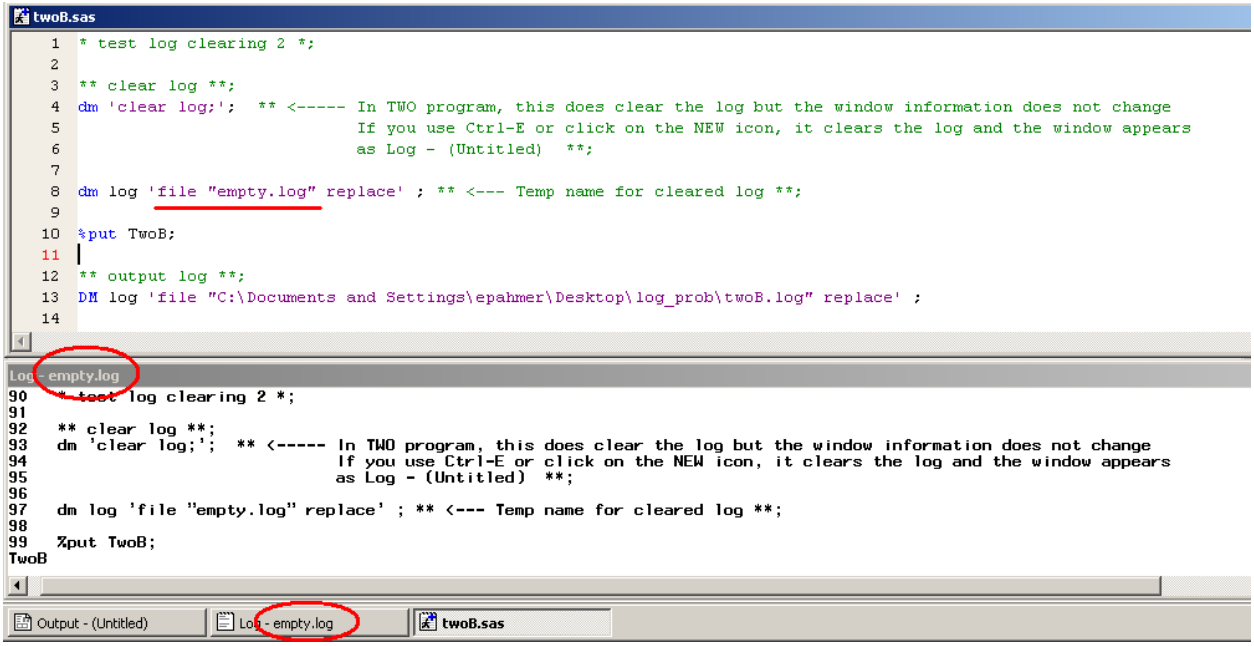
To manually clear the log, hit Ctrl-E or the 'New' icon when you are in the log window.

There is a problem with this. If we run one program and save the log, then run a second program which clears the log with a DM statement at the beginning, the name of the first log stays in the window as in Display 1 below. The log has been cleared but the name of the previous log still appears.



Display 1. Problem Clearing the Log with a DM Statement

There are a couple of solutions to this. We can clear the log manually, which changes the name to Log – (Untitled) or add a statement to give it another, temporary name. See Display 2 below. If the log is completely empty then the temporary log file will not be created. Whether or not the log is empty may depend on system options discussed later on.



Display 2. Redirect to a Temporary Log

Another solution proposed by SAS Support, was to close and re-open the log but that changes the order of the tabs (log, output and program) at the bottom.

SAVING

Logs should be saved to prove how and when the program was run. The log can be saved using either the PRINTTO procedure or a DM statement. PRINTTO will send future log information to the specified destination. So from that

moment on, whatever would normally go to the log window goes to that destination instead and your log window (if you're on PC SAS) will be empty. You would use it at the beginning of your program.

```
proc printto log = 'C:\my_log.txt' replace;
run;
```

Using a DM statement tells SAS to send whatever is already in the log window to the permanent file specified and is used at the end of the program.

```
dm log 'file "C:\my_log2.txt" replace' log;
```

If you're on a mainframe system, you don't have a log window so you would use PROC PRINTTO. If you're on PC, you have the choice.

LOG NAME

Often we wish for the log to have the same name as the program, with a different extension. We can get the program name from the SAS_EXECFILENAME environment variable:

```
%letpgname = %scan(%sysget(SAS_EXECFILENAME),1, '.');
```

However, if this program is being called by another, it's the higher-level program (ie the batch program name) which will be used.

CUSTOM NOTES, ERRORS AND WARNINGS

SAS automatically gives us notes, errors and warnings but we can add our own, and SAS will use the usual colour scheme to highlight them in the log. The words "NOTE", "WARNING" or "ERROR" have to be uppercase and followed by a colon or a dash:

```
data_null_;
    put'WARNING: Custom warning with colon';
    put'WARNING- Custom warning with dash';
run;
```

Log:

```
WARNING: Custom warning with colon
         Custom warning with dash
```

Here, the default settings have been changed to use Attribute=Reverse so that these messages really stand out.

Tools -> Options->Colors then select Error, and change the attribute to 'Reverse'.

If the source code appears in the log then these PUT statements will appear in the log checking report. A way to avoid this is to break up the word "warning":

```
Put 'WAR' 'NING: This put statement will not appear in log checking report';
```

Log:

```
4806   data _null_;
4807   put'WAR' 'NING: This put statement will not appear in log checking report';
4808   run;

WARNING: This put statement will not appear in log checking report
```

SYSTEM OPTIONS AFFECTING THE LOG

The main system options to consider for log messaging are SOURCE, SYMBOLGEN, MLOGIC and MPRINT.

- SOURCE: shows the source code in the log.
- SYMBOLGEN: shows how macro variables resolve
  `Eg. SYMBOLGEN: Macro variable I resolves to 1`
- MLOGIC: traces macro execution
  `Eg.MLOGIC(A_MACRO): %DO loop index variable I is now 3; loop will not iterate again.`
- MPRINT: displays the SAS statements generated by macro execution

Program:

```
%macro a_macro;

%do i = 1 %to 1;
   datads&i;
       total = &i + &i;
   run;
%end;

%mend;

%a_macro;
```

Log:

```
MPRINT(A_MACRO):   data ds1;
MPRINT(A_MACRO):   total = 1 + 1;
MPRINT(A_MACRO):   run;
```

To turn these features off, add “NO” to the beginning, eg. NOSOURCE. Different programmers might have different preferences regarding these options. If a programmer doesn’t want all the information that SAS provides with some of these options, and prefers to turn them off, then explicit statements should be written so that the log contains the appropriate information about processing. The options may also be turned on and off throughout the program, depending on what you want to display.

WHAT IS NOT PERMISSIBLE

Within your organization you have to decide what kinds of messages are permissible in the program logs. Of course, there should be no errors. Most of the time you don’t want warnings either, but there may be exceptions, like user-defined warnings for data issues. The same may be true for certain notes, like

```
NOTE: Numeric values have been converted to character values at the places
given by:
```

In these cases, another message can be added to the log stating that these messages are expected in certain cases. Some messages are definitely unacceptable but do not, by default, get considered errors by SAS, such as the infamous

```
NOTE: MERGE statement has more than one data set with repeats of BY values.
```

LOG CHECKING

Log checking should be automated for large production programs or in the case where you have several programs creating several logs.

Here's a sample program to check one log, combine questionable messages and output them. A real production program would be a little more elaborate, looping through several logs, combining the results and outputting a report to a permanent file.

```
%let logpath = C:\<etc>\log_prob;
%let filename = my_log.log;

       Data onelog;
              length line $200 logfile $12;
              infile "&logpath.\&filename" pad missover end = at_end;
              input line $ 1 - 200;
              line = upcase(line);
              retain flag 0;
              logfile = "&filename";
              if     (      line =: 'ERROR' or
                            line =: 'WARNING' or
                            index(line, 'REPEATS OF BY VALUES') or
                            index(line, 'UNINITIALIZED') or
                            index(line, 'FORMAT WAS TOO SMALL') or
                            index(line, 'HAVE BEEN CONVERTED TO') or
                            index(line, 'INVALID ARGUMENT') or
                            index(line, 'OPERATIONS COULD NOT BE PERFORMED') or
                            index(line, 'STOPPED PROCESSING BECAUSE OF ERRORS')
                     )
                     and not index(line, 'NO MATCHING MEMBERS IN DIRECTORY') then
              do;
                     flag = 1;
                     output;
              end;
              else if at_end and flag = 0 then
              do;
                     line = '(none)';
                     output;
              end;
       run;

title "&filename Errors, warnings, special notes in log.  ";
proc report data = onelog nowindows;
       columns logfile line;
       define logfile / 'Log File'order  ;
       define line    / 'Line from log' displaywidth=97 flow;
run;
```

Read the lines of the log into SAS as data

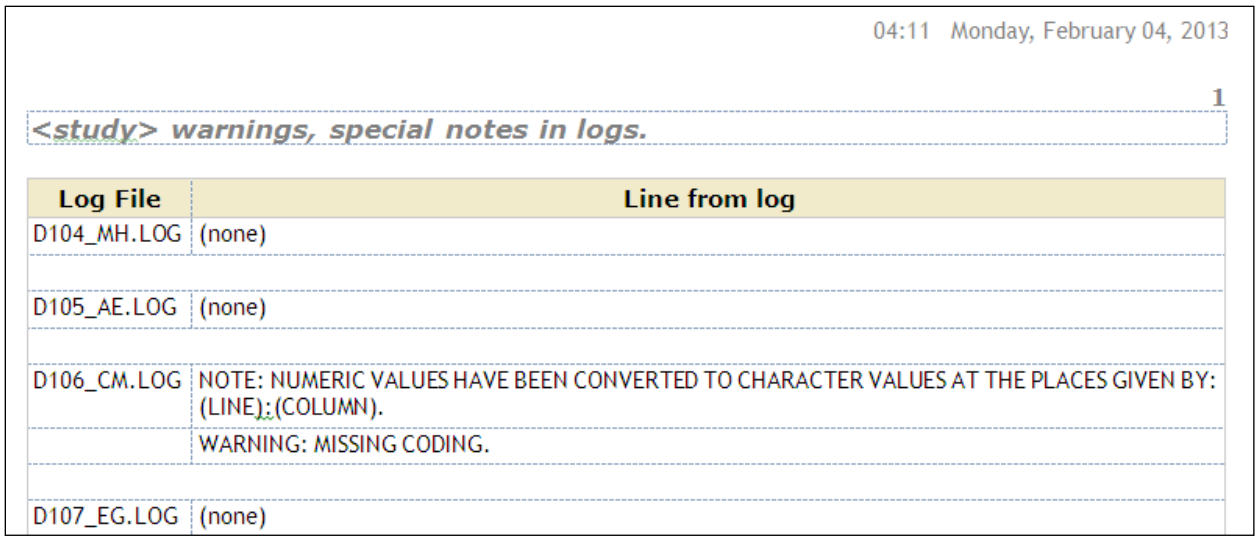
Check for these messages

Output questionable messages…

…or indicate that there are none

Print them

A log report that checks several logs may look something like Figure 1 below. The main purpose is to identify which individual programs need to be examined. In the end, the report should contain "(none)" for all program logs.



04:11 Monday, February 04, 2013

1

<study> warnings, special notes in logs.

Log File	Line from log
D104_MH.LOG	(none)
D105_AE.LOG	(none)
D106_CM.LOG	NOTE: NUMERIC VALUES HAVE BEEN CONVERTED TO CHARACTER VALUES AT THE PLACES GIVEN BY: (LINE):(COLUMN).
	WARNING: MISSING CODING.
D107_EG.LOG	(none)

Figure 1. Sample log checking report

COMPLETE AND FINAL LOG

Ensure that the log contains information from the most recent version of your program. Save the program before running it. The log should have a time stamp after the program's. If you're running a program bit by bit to debug, then, once it has been corrected, run the whole thing to have a complete, final and clean log.

CHANGES TO DATA

Often decisions are made as to how data needs to be converted or formatted. Sometimes this is part of standard processing, or it may be an exceptional case. If it is an exceptional case and the decision is supported by sufficient documentation, data changes may be made and having a note in the log is useful.

Eg. Program:

```
data a ;
    setsashelp.class;
    if name = 'Alfred' and weight = 112.5 then
    do;
           put / 'For this subject, weight was entered incorrectly and unable to
           change in data capture system.'
                  / 'See Note to File #1 dated 01MAR2014.  Changing weight value:'
                  / name= ;
           Put 'Before: ' weight=;
           weight = 120;
           put 'After:  ' weight=;
    end;
run;
```

Log:

```
For this subject, weight was entered incorrectly and unable to change in data
capture system.
See Note to File #1 dated 01MAR2014.  Changing weight value:
Name=Alfred
Before: Weight=112.5
After:  Weight=120
```

If you have more than one type of change, you might want to consider two different DATA steps to have a cleaner looking log.

In one DATA step:

```
data class2;
      setsashelp.class;
      if name in ('John','Robert') then /* change 1 */
      do;
            put 'Deleting this student, dropped out. ' name=;
            delete;
      end;
      if name in ('Judy','Thomas') then  /* change 2 */
      do;
            put 'Correcting weight. ' name=  @32 'Before ' weight= ;
            weight = weight * 1.1;
            put @32 'After ' weight=;
      end;
run;
```

Log:

```
Deleting this student, dropped out. Name=John
Correcting weight. Name=Judy   Before Weight=90
                              After Weight=99
Deleting this student, dropped out. Name=Robert
Correcting weight. Name=Thomas Before Weight=85
                              After Weight=93.5
```

In 2 steps:

```
data class3;
      setsashelp.class;
      if name in ('John','Robert') then /* change 1 */
      do;
            put 'Deleting this student, dropped out. ' name=;
            delete;
      end;
run;

data class4;
      set class3;
      if name in ('Judy','Thomas') then   /* change 2 */
      do;
            put 'Correcting weight. ' name=  @32 'Before ' weight= ;
            weight = weight * 1.1;
            put @32 'After ' weight=;
      end;
run;
```

Log:

```
Deleting this student, dropped out. Name=John
Deleting this student, dropped out. Name=Robert

 (...)

Correcting weight. Name=Judy   Before Weight=90
                               After Weight=99
Correcting weight. Name=Thomas Before Weight=85
                               After Weight=93.5
```

Sometimes we run code several times in order to make a correction. In this case it's a lot easier to write a line of code to make sure you are getting the correct outcome rather than to open up the resulting dataset after each run, manually select the required observations and variables and do a visual check.

INTRO AND OUTRO

Different systems provide different information by default. You might want to put certain user and system information in your log, and information about when the program was run. Here's an example (see the Log Name section earlier):

```
%put NOTE- ************************************************************;
%put NOTE-   Program %upcase(&pgname.).SAS started at %sysfunc(datetime(), datetime.);
%put NOTE-   Run by: &sysuserid;
%put NOTE-   SAS Version: &sysvlong;
%put NOTE-   OS: &sysscp&sysscpl;
%put NOTE-   Computer: &systcpiphostname;
%put NOTE- ************************************************************;
```

Log:

```
************************************************************
  Program TEST_PGM.SAS started at 08MAR14:14:08:26
  Run by: epahmer
  SAS Version: 9.02.02M3P041310
  OS: WIN XP_PRO
  Computer: XXXX00000
************************************************************
```

To get only this text in the log, the NOSOURCE and NOSYMBOLGEN options must be used. Similarly, information regarding when the program terminated might be useful at the end of the program.

CHEATING

Sometimes we use procedures to check data and would like to send the output to the log, especially when output files are not saved but log files are. (To my knowledge, DATSETS is the only procedure to send some of the results to the log.) It would be nice if it were as simple as using PROC PRINTTO, but that doesn't work. The way to get around this is by saving the output to a data set and using PUT statements to get it into the log. If it's a task you perform often, it's worth writing a macro, such as in the case of PROC CONTENTS:

```
%macro contents_to_log(ln=, ds= );

** OUTPUT PROC CONTENTS TO DATASET **;
proc contents data = &ln..&ds out=proc_contents noprint varnum; run;
proc sort data = proc_contents; by varnum; run;
options ls = 140;

** GET PATH **;
proc sql;
       select path into : path
       from sashelp.vslib
       where upcase(libname) = upcase("&ln");
quit;
%let path = %sysfunc(strip(&path));

** GET SORT VARIABLES **;
%let sv = ;  ** create and set as empty in case there are none **;
proc sql;
       select name into :sv separated by ', '
       from proc_contents (where = (sortedby ne .))
       order by sortedby;
quit;

** PRINT EVERYTHING **;
data _null_;
       set proc_contents end = at_end;
       if type = 1 then typec = 'Num ';
```

```
      else if type = 2 then typec = 'Char';
      format = cats(format, formatl);
      informat = cats(informat, informl);
      if format = '0' then format = ' ';
      if informat = '0' then informat = ' ';
      if _n_ = 1 then
      do;
            put 140 * '=';
            put "CONTENTS of &ln..&ds";
            put 140 * '=';
            put / libname= / memname= / "PATH=&path" / memlabel= / nobs= / modate= /
crdate= ;
            put / 'VARNUM' @10 'NAME' @46 'LABEL'  @90 'TYPE' @100  'LENGTH' @110
'FORMAT' @120 'INFORMAT' ;
      end;
      put varnum @10 name @46 label @90 typec @100 length @110 format @120 informat;
      if at_end then
      do;
            put / "Sort variables: &sv";
            put / 140 * '=';
      end;
run;

%mend;
```

In the log

Example:

```
data mydata (label = 'Example of Output to Log');
    attrib  var1 label = 'First Variable' length = $10
            var2 label = 'Second Variable' length = $20
            var3 label = 'Third Variable' length = 8;
    var1 = 'a';
    var2 = 'b';
    var3 = 1;
run;
proc sort data = mydata; by var1 var2; run;

%contents_to_log(ln = work, ds = mydata);
```

Create sample data

Send the PROC CONTENTS to the log

Log:

```
================================================================================================
CONTENTS of work.mydata
================================================================================================
LIBNAME=WORK
MEMNAME=MYDATA
PATH=C:\DOCUME~1\epahmer\LOCALS~1\Temp\1c\SAS Temporary Files\_TD3728
MEMLABEL=Example of Output to Log
NOBS=1
MODATE=17FEB14:18:33:57
CRDATE=17FEB14:18:33:57

VARNUM   NAME                  LABEL                     TYPE      LENGTH    FORMAT    INFORMAT
1        var1                  First Variable            Char      10
2        var2                  Second Variable           Char      20
3        var3                  Third Variable            Num       8

Sort variables: var1, var2

================================================================================================
```

CONCLUSION

Keeping the log in mind from the moment you start programming is well worth the effort in order to end up with a document that illustrates that the program ran error-free. It will also contain all other pertinent documentation to illustrate what is happening in the program or to the data.

ACKNOWLEDGMENTS

Thanks to André Nadeau and Nancy Brucken.

RECOMMENDED READING

For more information on the log, go to http://support.sas.com/ and search "The SAS Log". Here is the current link:

http://support.sas.com/documentation/cdl/en/lrcon/65287/HTML/default/viewer.htm#n03qoiyzzrrl4in1pfvbqgj7jan8.htm

CONTACT INFORMATION

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

Name: Emmy Pahmer
Enterprise: inVentiv Health Clinical
E-mail: emmy.pahmer@inventivhealth.com