SESUG Paper 96-2024

Log Reviewing Made Easy

Peter Knapp, U.S. Department of Commerce

ABSTRACT

One of the golden rules when running SAS® programs is to review the Log for problems. Unfortunately, many users ignore the Log and focus on the output. Adding a dynamically generated report to the bottom of the Log that shows the number of problems (such as run-time errors, missing values, and uninitialized variables) is a great way to identify issues with the program run that need addressing. Enhancing the report to include information about the number of observations flowing through the program can not only help ensure that the program runs without mechanical errors, but also runs as expected. This presentation discusses a Log Report macro that provides summary log information, explains how it works, and details how to modify it so you can tailor it for your organization's specific needs. I developed this Log Report using SAS® Enterprise Guide® on a Windows environment using SAS macro and Base SAS code. The Log Report also works in SAS® Studio.

INTRODUCTION

I support 150 trade analysts who use SAS to administer international trade law. Most of the analysts have no programming background yet need to customize large template programs with case specific information. They find reviewing the thousands of lines of the Log for problems difficult and confusing.

I developed a Log Report macro that summarizes problems with program runs and provides information about the flow of data. This allows the analysts to quickly determine if their programs have run properly and helps them find code that needs revision. By adding a minimal amount of Base SAS and SAS macro a program, the Log Report can speed up the debugging process as it identifies problems that need review and correction.

For this paper I am using a simplified Log Report that will be reviewing a small program. The Log Report can easily be expanded to capture more indicators of problems with the run program run along with information about the data flow.

REVIEWING THE LOG FOR PROGRAM RUN PROBLEMS

It's important to make sure that your program has run correctly. Using the Log Summary built into SAS Enterprise Guide it's easy to find Syntax Errors and Warnings.

Finding other issues explained in the notes can be more challenging. Examples include:

- Converted variables
- Division by zero detected
- Invalid data values
- Missing values
- Rejected missing weight variable values
- Repeats of by variables
- Uninitialized variables

SAMPLE PROGRAM

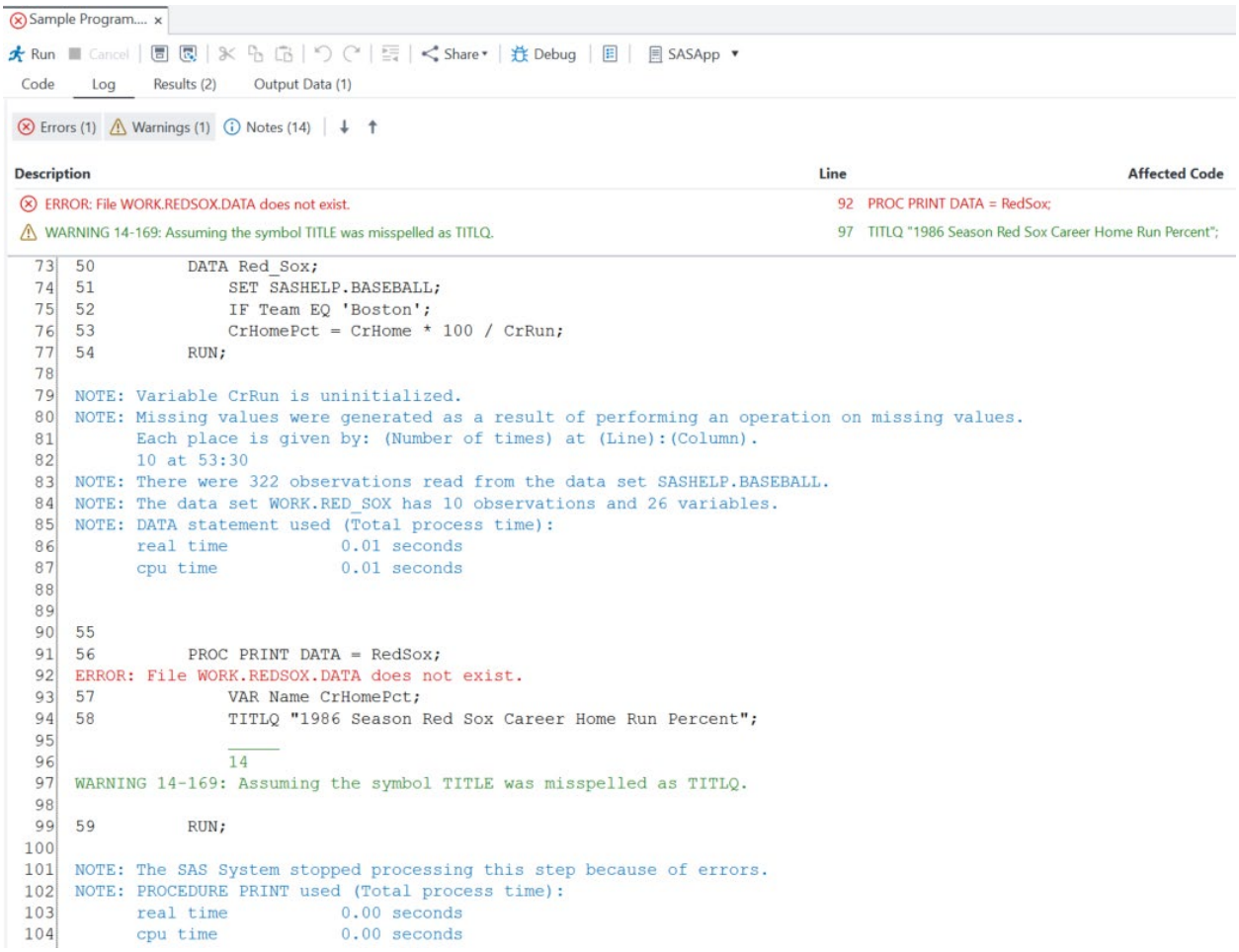
When SAS runs the following example program:

```
DATA Red_Sox;
     SET SASHELP.BASEBALL;
     IF Team EQ 'Boston';
     CrHomePct = CrHome * 100 / CrRun;
```

```
RUN;

PROC PRINT DATA = RedSox;
     VAR Name CrHomePct;
     TITLQ "1986 Season Red Sox Career Home Run Percent";
RUN;
```

it generates an uninitialized variable NOTE, a run time ERROR, and WARNING message:



Display 1. Sample Log in SAS Enterprise Guide

In the Log Summary the Errors tab indicates the total number of syntax and run time errors. The Description section lists the error messages, and the ERROR messages appear in the Log. Clicking on an ERROR message in the Description section will highlight the error listed in the Log.

Similarly, the Warnings tab indicates the total number of warning messages. The Description section lists the warning messages, and the WARNING messages appear in the Log. Clicking on a WARNING message in the Description section will highlight the warning listed in the Log.

Notes behave the same way in the Log Summary. Because the Log Summary identifies and lists all the notes, it can be difficult to know which notes are indicative of problems with the program run.

THE LOG REPORT IS DIFFERENT THAN THE LOG SUMMARY

The Log Summary tabulates all notes together. You need to review all notes to find notes indicative of problems with the program run. The Log Report Identifies specific kinds of Notes indicative of problems with the program run. There is no need to review all notes. The report can be enhanced to include information about the number of observations flowing through the program.

IMPLEMENTING THE LOG REPORT

To implement the Log Report, you need to add the following functionality into your program:

1. Redirect the Log to an external file.
2. Run the program to be reviewed.
3. Reset the Log Destination to the Log.
4. Run the Log Report macro.

1. REDIRECT THE LOG TO AN EXTERNAL FILE

The first step is to use a FILENAME Statement to define the external file path and file name. The following is an example of a FILENAME statement:

```
FILENAME FILEREF "C:\Sample Program.log";
```

Alternatively, you can dynamically define the external file with the same path and name of SAS program to be reviewed by the Log Report, though with a 'LOG' suffix instead of a 'SAS' suffix. Here is an example of a FILENAME statement that dynamically generates the path and file name:

```
FILENAME FILEREF "%SYSFUNC(TRANWRD(%UPCASE(&_SASPROGRAMFILE), .SAS, .LOG))";
```

The automatic macro variable _SASPROGRAMFILE is used to identify the path and name of the SAS program. The %SYSFUNC macro function executes SAS functions or user-written functions. The TRANWRD macro function replaces all occurrences of a substring in a character string. The %UPCASE macro function converts values to uppercase.

After the FILENAME is defined SAS needs to be directed to send the log destination to an external file. Here is an example of a PROC PRINTO which defines destinations, other than ODS destinations, for SAS procedure output and for the SAS log:

```
PROC PRINTTO LOG = LOG;
RUN;
```

2. RUN THE PROGRAM TO BE REVIEWED

The second step is to run the program. The Log of the program run will be sent to the external file C:\Sample Program.log. Here is the sample program:

```
DATA Red_Sox;
    SET SASHELP.BASEBALL;
    IF Team EQ 'Boston';
```

```
    CrHomePct = CrHome * 100 / CrRun;
RUN;

PROC PRINT DATA = RedSox;
    VAR Name CrHomePct;
    TITLQ "1986 Season Red Sox Career Home Run Percent";
RUN;
```

3. RESET THE LOG DESTINATION TO THE LOG

The third step is to reset the log destination back to the default log destination. Here is an example of a PROC PRINTO setting the log destination to the SAS Log:

```
PROC PRINTTO LOG = LOG;
RUN;
```

4. RUN THE LOG REPORT MACRO

The fourth step is to run the Log Report macro which does six things:

i. Copies the saved Log to the default log destination.
ii. Reads the saved Log.
iii. Looks for key words.
iv. Accumulates instances of key words.
v. Saves key word totals into macro variables.
vi. Writes key word totals to the Log Report.

4.i. Copy the Saved Log to the Default Log Destination

The %MACRO statement defines the beginning of the macro LOG_REPORT:

```
%MACRO LOG_REPORT;
    DATA _NULL_;
        INFILE FILEREF;
        INPUT;
        PUTLOG _INFILE_;
    RUN;
```

The DATA _NULL_ uses the following statements and keywords. The INFILE statement specifies an external file to read with an INPUT statement. The INPUT statement without arguments brings an input data record into the input buffer without creating any SAS variables. The PUTLOG statement writes a message to the SAS log. The _INFILE_ automatic variable contains the value of the current input record read from a file.

4.ii. Read the Saved Log

The DATA _NULL_ reads each line of the log:

```
 DATA _NULL_;
    INFILE FILEREF END = END MISSOVER PAD;
    INPUT LINE $250.;
```

The END = option specifies a variable that SAS sets to 1 when the current input data record is the last in the input file. The MISSOVER option prevents an INPUT statement from reading a new input data record if it does not find values in the current input line for all the variables in the statement. The PAD option pads the records that are read from an external file with blanks. The INPUT statement reads each line of the Log into the variable LINE.

4.iii. and 4.iv. Look for Key Words and Accumulate Instances of Key Words

The conditional logic processes each line of the Log and accumulates key word counters when it finds instances of keywords:

```
IF UPCASE(COMPRESS(SUBSTR(LINE,1,6))) = "ERROR:" THEN
    ERROR + 1;
ELSE IF UPCASE(COMPRESS(SUBSTR(LINE,1,8))) = "WARNING:" THEN
    WARNING + 1;
ELSE DO;
    UNINIT_I = INDEX(UPCASE(LINE),'UNINITIALIZED');
    IF UNINIT_I THEN
        UNINIT + 1;
END;
```

When looking for the keyword ‘ERROR:’ the code uses UPPER, COMPRESS, and SUBTR functions to check the first six digits of the variable LINE. When ‘ERROR:’ is found the code accumulates the variable ERROR. The same three functions are used to check the first eight digits of the variable LINE to look for the keyword ‘WARNING:’. When found the WARNING accumulator is increased. To look for the keyword ‘UNINITIALIZED’ the INDEX and UPCASE functions are used. When found UNINIT accumulator is increased.

4.v. Save Key Word Totals into Macro Variables

The CALL SYMPUTX routine is used to assign a value to a macro variable and remove both leading and trailing blanks. Macro variables created by CALL SYMPUTX are not available until after the DATA Step is run. The CALL SYMPUTX routine looks like this:

```
    CALL SYMPUTX('ERROR', ERROR);
    CALL SYMPUTX('WARNING', WARNING);
    CALL SYMPUTX('UNINIT', UNINIT);
RUN;
```

4.vi. Write Key Word Totals to the Log

The %PUT statements to write the Log Report to the SAS Log. The %MEND statement. ends the macro definition:

```
    %PUT ************************************************************;
    %PUT * GENERAL SAS ALERTS: Determine cause of non-zero instances. *;
    %PUT ************************************************************;
    %PUT # OF ERRORS                  = &ERROR;
    %PUT # OF WARNINGS                = &WARNING;
    %PUT # OF UNINITIALIZED VARIABLES = &UNINIT;
%MEND LOG_REPORT;

%LOG_REPORT
```

RESULTS OF LOG REPORT MACRO RUN

The following Log shows the Log Report macro run. Note: %PUT statements don't appear in the Log:

```
92            %LOG_REPORT
MPRINT(LOG_REPORT):   DATA _NULL_;
MPRINT(LOG_REPORT):   INFILE FILEREF;
MPRINT(LOG_REPORT):   INPUT;
MPRINT(LOG_REPORT):   PUTLOG _INFILE_;
MPRINT(LOG_REPORT):   RUN;

NOTE: The infile FILEREF is:
      Filename=E:\OPERATIONS\ADCVDTR\PETER\SAMPLE PROGRAM.LOG,
      RECFM=V,LRECL=32767,File Size (bytes)=1625,
      Last Modified=23Aug2024:23:55:15,
      Create Time=23Aug2024:23:38:45

43
44            DATA Red_Sox;
45                SET SASHELP.BASEBALL;
46                IF Team EQ 'Boston';
47                CrHomePct = CrHome * 100 / CrRun;
48            RUN;

NOTE: Variable CrRun is uninitialized.
NOTE: Missing values were generated as a result of performing an operation on missing values.
      Each place is given by: (Number of times) at (Line):(Column).
      10 at 47:33
NOTE: There were 322 observations read from the data set SASHELP.BASEBALL.
NOTE: The data set WORK.RED_SOX has 10 observations and 26 variables.

49
50            PROC PRINT DATA = RedSox;
ERROR: File WORK.REDSOX.DATA does not exist.
51                VAR Name CrHomePct;
52                TITLQ "1986 Season Red Sox Career Home Run Percent";
                  _____
                  14
WARNING 14-169: Assuming the symbol TITLE was misspelled as TITLQ.

53            RUN;

NOTE: The SAS System stopped processing this step because of errors.
54
55               PROC PRINTTO LOG = LOG;
56            RUN;
NOTE: 47 records were read from the infile FILEREF.
      The minimum record length was 0.
      The maximum record length was 133.

MPRINT(LOG_REPORT):   DATA _NULL_;
MPRINT(LOG_REPORT):   INFILE FILEREF END = END MISSOVER PAD;
MPRINT(LOG_REPORT):   INPUT LINE $250.;
MPRINT(LOG_REPORT):   IF UPCASE(COMPRESS(SUBSTR(LINE, 1, 5))) = "ERROR" THEN ERROR + 1;
MPRINT(LOG_REPORT):   ELSE IF UPCASE(COMPRESS(SUBSTR(LINE, 1, 7))) = "WARNING" THEN WARNING + 1;
MPRINT(LOG_REPORT):   ELSE DO;
MPRINT(LOG_REPORT):   UNINIT_I = INDEX(UPCASE(LINE), 'UNINITIALIZED');
MPRINT(LOG_REPORT):   IF UNINIT_I THEN UNINIT + 1;
MPRINT(LOG_REPORT):   END;
MPRINT(LOG_REPORT):   CALL SYMPUTX('ERROR', ERROR);
MPRINT(LOG_REPORT):   CALL SYMPUTX('WARNING', WARNING);
MPRINT(LOG_REPORT):   CALL SYMPUTX('UNINIT', UNINIT);
MPRINT(LOG_REPORT):   RUN;

NOTE: The infile FILEREF is:
      Filename=E:\OPERATIONS\ADCVDTR\PETER\SAMPLE PROGRAM.LOG,
      RECFM=V,LRECL=32767,File Size (bytes)=1625,
      Last Modified=23Aug2024:23:55:15,
```

```
      Create Time=23Aug2024:23:38:45

NOTE: 47 records were read from the infile FILEREF.
      The minimum record length was 0.
      The maximum record length was 133.

************************************************************
* GENERAL SAS ALERTS: Determine cause of non-zero instances. *
************************************************************
# OF ERRORS                  = 1
# OF WARNINGS                = 1
# OF UNINITIALIZED VARIABLES = 1
```

ENHANCING THE LOG REPORT MACRO FOR YOUR SPECIFIC NEEDS

Anything appearing in the Log can be identified and added to the Log Report Macro. For example, the Log Report Macro can identify how many observations are read in from a data set and how many observations are kept. The following program reads the BASEBALL dataset from the SASHELP library and keeps Red Sox players:

```
DATA Red_Sox;
     SET SASHELP.BASEBALL;
     IF Team EQ 'Boston';
RUN;
```

ADDING FUNCTIONALITY TO THE LOG REPORT MACRO

Because anything the appears in the Log can be identified, the Log Report can be enhanced to display more than instances of ERRORs, WARNINGs, and NOTEs.

Identify and Print Data Set Observations

The PROC SQL SELECTs the total number of observations FROM a specified data set and assigns totals INTO macro variables. %PUT statements write messages to the SAS Log:

```
PROC SQL NOPRINT;
     SELECT COUNT(*)
     INTO :COUNT_CARS
     FROM SASHELP.BASEBALL;
QUIT;

PROC SQL NOPRINT;
     SELECT COUNT(*)
     INTO :COUNT_SAVINGS
     FROM WORK.Red_Sox;
QUIT;

%PUT ************************************************************;
%PUT * PROGRAM FLOW OF DATA IN THE PROGRAM: Obs before and after. *;
%PUT ************************************************************;
%PUT # OF PLAYERS IN SASHELP.BASEBALL  = %CMPRES(&COUNT_CARS);
%PUT # OF RED SOX IN WORK.Red_Sox      = %CMPRES(&COUNT_SAVINGS);
```

Results of the Enhanced Log Report Macro Run

The following Log shows the program run:

- SELECTs total number of observations FROM specified data sets and assigns totals INTO macro variables.
- Writes key word totals to the Log Report using %PUT statements. Note: %PUT statements don't appear in the Log:

```
MPRINT(LOG_REPORT):   PROC SQL NOPRINT;
MPRINT(LOG_REPORT):   SELECT COUNT(*) INTO :COUNT_CARS FROM SASHELP.BASEBALL;
MPRINT(LOG_REPORT):   QUIT;
NOTE: PROCEDURE SQL used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds


MPRINT(LOG_REPORT):   PROC SQL NOPRINT;
MPRINT(LOG_REPORT):   SELECT COUNT(*) INTO :COUNT_SAVINGS FROM WORK.Red_Sox;
MPRINT(LOG_REPORT):   QUIT;
NOTE: PROCEDURE SQL used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds


************************************************************
* PROGRAM FLOW OF DATA IN THE PROGRAM: Obs before and after. *
************************************************************
# OF PLAYERS IN SASHELP.BASEBALL  = 322
# OF RED SOX IN WORK.Red_Sox      = 10
```

SETTING UNINITIALIZED VARIABLES NOTES TO ERROR MESSAGES

When SAS tries to use a variable that doesn't exist, it issues a NOTE in the Log and continues to compile and run the rest of the program. For example, in this Log the variable CrRuns has been misspelled as CrRun:

```
39            DATA Red_Sox;
40                SET SASHELP.BASEBALL;
41                IF Team EQ 'Boston';
42                CrHomePct = CrHome * 100 / CrRun;
43            RUN;

NOTE: Variable CrRun is uninitialized.
NOTE: Missing values were generated as a result of performing an operation on missing values.
      Each place is given by: (Number of times) at (Line):(Column).
      10 at 42:33
NOTE: There were 322 observations read from the data set SASHELP.BASEBALL.
NOTE: The data set WORK.RED_SOX has 10 observations and 26 variables.
```

SAS initializes the variable CrRun and sets the value to missing. When CrRun is used in a calculation the result CrHomePct is also set to missing. The NOTES indicating uninitialized variables and missing values, that can be indicative of undesired results, can be hard miss when visually scanning the Log. The option VARINITCHK can be set to interpret uninitialized variables as ERRORs as shown in this Log:

```
37            OPTION VARINITCHK = ERROR;
38
```

```
39          DATA Red_Sox;
40              SET SASHELP.BASEBALL;
41              IF Team EQ 'Boston';
42              CrHomePct = CrHome * 100 / CrRun;
43          RUN;

ERROR: Variable CrRun is uninitialized.
NOTE: The SAS System stopped processing this step because of errors.
WARNING: The data set WORK.RED_SOX may be incomplete.  When this step was stopped
there were 0 observations and 26 variables.
WARNING: Data set WORK.RED_SOX was not replaced because this step was stopped.
```

When reviewing the Log, it is much easier to interpret ERRORs that indicate problems with the program run than to interpret NOTEs that indicate problems with the program run. The use of the VARINITCHK option makes it a lot easier to find uninitialized variables that lead to missing values and often unwanted results.

CONCLUSION

There are many ways to ensure that programs run properly. The Log Summary is a great way to identify ERRORs and WARNINGs but not NOTEs indicative of problems. A Log Report, in addition to identifying ERRORs and WARNINGs, identifies specific kinds of NOTEs indicative of problems with the program run. The Log Report can be enhanced to identify anything the appears in the Log Summary.

A Log Report is especially helpful when running programs that generate thousands of lines in the Log that are time intensive to review manually. With the Log Report you can focus on correcting any problems with your program and producing the results you want.

REFERENCES

SAS® 9.4 and SAS® Viya® 3.5 Programming Documentation. Available at https://documentation.sas.com/doc/en/pgmsascdc/9.4_3.5/pgmsashome/home.htm

Knapp, Peter. 2021. “Log Reviewing Made Easy.” Proceedings of the Virtual SAS Global Forum 2021. Available at https://communities.sas.com/t5/SAS-Global-Forum-Proceedings/Log-Reviewing-Made-Easy/ta-p/726304 and https://www.youtube.com/watch?v=-cOsjd6j40U&t=1223s.

ACKNOWLEDGMENTS

This is the I would like to thank my colleague Girish Narayandas who adapted and improved the Log Report macro I originally wrote in Base SAS to work with SAS Enterprise Guide.

CONTACT INFORMATION

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

Peter Knapp
U.S. Department of Commerce
202-482-1359
Peter.Knapp@trade.gov
www.linkedin.com/in/peter-knapp-aa752890