SESUG 2020 Paper 112

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 in one place issues with the program run that need addressing. Enhancing the report to include information about the number of observations flowing through the program can help insure not only that the program runs without mechanical errors, but also runs as expected. In this paper I will talk about a Log Report macro that provides that summary log information, explain how it works, and discuss how to modify it so that you can tailor it for your organization's specific needs. The Log Report was developed using SAS Enterprise Guide on a Windows environment using SAS macro code in addition to Base SAS Code.

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 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. Use of 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 along with more information about the data flow.

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. Copy the Log from the external file to the SAS Enterprise Guide Log window.
4. Run the LOG_REPORT macro.

REDIRECT THE LOG TO AN EXTERNAL FILE

The following code redirects the Log to an external file:

```
1 %LET LOG_FILE = E:\Operations\ADCVDTR\Peter\Log_File.log;
2
3 FILENAME LOGFILE "&LOG";
4
5 PROC PRINTTO LOG = LOGFILE NEW;
6 RUN;
```

This is accomplished by the following actions:

1. At line 1 the macro variable LOG_FILE is assigned the external file location to write the Log to.
2. At line 3 the FILENAME statement assigns the macro variable LOG_FILE to the fileref LOGFILE.
3. At line 5/6 the PROC PRINTTO redirects the Log to the file 'Log_File.log'.

RUN THE PROGRAM TO BE REVIEWED

This short program finds the difference between MSRP and INVOICE for cars made in the USA. It uses the SASHELP.CARS data set:

```
 7 DATA CARS;
 8     SET SASHELP.CAR;
 9     IF ORIGIN EQ 'USA';
10     DISCOUNT = MSRP - INVOIC;
11 RUN;
12
13 PROC PRINT DATA = CAR;
14 RUN;
```

There are two problems with the program:

1. At line 10 the variable INVOICE is misspelled as INVOIC.
2. At line 13 the temporary data set CARS is misspelled as CAR.

COPY THE SAVED LOG FILE TO THE SAS ENTERPRISE GUIDE LOG WINDOW

The PROC PRINTTO prints the copies the Log from the external file to the SAS Enterprise Guide Log window:

```
15 PROC PRINTTO LOG = LOG;
16 RUN;
```

RUN THE LOG_REPORT MACRO

The macro variable LOG_FILE passes the location and name of the text file containing the Log to the LOG_REPORT macro:

```
17 %LOG_REPORT (LOG = &LOG_FILE);
```

WHAT THE LOG_REPORT MACRO DOES

The LOG_REPORT macro, does the following:

1. Writes the Log to the SAS enterprise guide Log tab.
2. Searches for terms that identify problems with the program run, keeping a count of each kind of identified problem.
3. Tracks the flow of data by capturing the number of observations in each data set.
4. Prints the Log Report at the bottom of the Log.

WRITE THE LOG TO THE SAS ENTERPRISE GUIDE LOG TAB

The DATA step reads in the Log from the external file into the data set _NULL_ and sends it to the Enterprise Guide Log tab:

```
1 %MACRO LOG_REPORT (LOG_FILE = );
2     DATA _NULL_;
3         INFILE LOGFILE;
4         INPUT;
5         PUTLOG _INFILE_;
6      RUN;
```

This is accomplished by the following actions:

1. At line 3 the INFILE statement specifies the external file to read with an INPUT statement.
2. At line 4 the INPUT statement reads the external file containing the Log of the program run.
3. At line 5 the PUTLOG statement writes the external file to the SAS Enterprise Guide Log window.

SEARCH THE LOG FOR TERMS THAT IDENTIFY PROBLEMS

The DATA step reads in the Log from the external file and searches for terms that identify problems with the program run, keeping a count of each kind of identified problem:

```
 7     DATA _NULL_;
 8         INFILE "&LOG_FILE" END = END MISSOVER PAD;
 9         INPUT LINE $250.;
10
11         IF UPCASE(COMPRESS(SUBSTR(LINE,1,6))) = "ERROR:"  THEN
12            ERROR + 1;
13         ELSE IF UPCASE(COMPRESS(SUBSTR(LINE,1,8))) = "WARNING:" THEN
14            WARNING + 1;
15         ELSE
16         DO;
17             UNINIT_I = INDEX(UPCASE(LINE),'UNINITIALIZED');
18             IF UNINIT_I THEN
19                 UNINIT + 1;
20             MISSING_I = INDEX(UPCASE(LINE), 'MISSING VALUES WERE');
21             IF MISSING_I THEN
22                 MISSING + 1;
23         END;
24
25         CALL SYMPUTX('ERROR', ERROR);
26         CALL SYMPUTX('WARNING', WARNING);
27         CALL SYMPUTX('UNINIT', UNINIT);
28         CALL SYMPUTX('MISSING', MISSING);
29     RUN;
```

This is accomplished by the following actions:

1. At lines 8/9 the external file containing the Log of the program run is read in.
2. At lines 11/12 the IF statement checks to see if data being read contains the term ‘ERROR:’. If the line contains the term ‘ERROR:’ the accumulator variable ERROR is incremented.

3. At lines 13/23 similar checks are being done for the terms ‘WARNING:’, ‘UNINITIALIZED’, and ‘MISSING VALUES WERE’. If these terms are found, their respective accumulator variables are incremented.
4. At lines 25/28 the CALL SYMPUTXs use the accumulator variables to create macro variables that will be used to print the Log Report.

ENHANCING THE LOG REPORT WITH ADDITIONAL TERMS

The Log Report in my organization also looks for the following terms that can indicate there are problems with the program run: ‘REPEAT’, ‘CONVERTED’, ‘DIVISION’, and ‘INVALID’. If there is a term you wish to have the Log Report modify the DATA step to do the following:

1. Expand the conditional processing to look for that term.
2. Increment an accumulator variable when that term if found.
3. Assign the value of the accumulator variable to a macro variable.
4. Print the macro variable in the Log Report.

TRACK THE FLOW OF DATA

To track the flow of data through the program capture the number of observations in the data sets used in the program:

```
30     PROC SQL NOPRINT;
31         SELECT COUNT(*)
32             INTO :COUNT_CARS
33                 FROM SASHELP.CARS;
34     QUIT;
35
36     PROC SQL NOPRINT;
37         SELECT COUNT(*)
38             INTO :COUNT_SAVINGS
39                 FROM WORK.CARS;
40     QUIT;
```

This is accomplished by the following actions:

1. At lines 30/34 the PROC SQL defines the macro variable COUNT_CARS and assigns it the number of observations in the data set SASHELP.CARS.
2. At lines 36/40 the number of observations in the data set WORK.CARS are counted and assigned to the macro variable COUNT_SAVINGS.

ENHANCING THE LOG REPORT WITH ADDITIONAL DATA SET TRACKING

The Log Report in my organization shows the number of observations for many permanent and temporary data sets. This is especially helpful when reviewing the results of merging data sets together as the Log Report shows the number of observations in the data sets being merged and the number of observations in the data set created by the merge. For each data set you want to track:

1. Add a PROC SQL to create a macro variable with the observation count of the dataset
2. Print the value of the macro variable in the Log Report.

PRINT THE LOG REPORT

At lines 41/56 multiple %PUT statements write the Log Report to the Log using the macro variables created above:

```
41    %PUT ************************************************************;
42    %PUT * GENERAL SAS ALERTS:                                        *;
43    %PUT ************************************************************;
44    %PUT *   NORMALLY, BELOW ALERTS SHOULD BE ZERO                    *;
45    %PUT *   IF THEY DO NOT HAVE ZERO INSTANCES                       *;
46    %PUT *   DETERMINE IF THERE IS AN ISSUE.                          *;
47    %PUT ************************************************************;
48    %PUT # OF ERRORS                        = &ERROR;
49    %PUT # OF WARNINGS                      = &WARNING;
50    %PUT # OF UNINITIALIZED VARIABLES       = &UNINIT;
51    %PUT # OF MISSING VALUES                = &MISSING;
52    %PUT ************************************************************;
53    %PUT *   THIS SECTION SHOWS THE FLOW OF DATA IN THE PROGRAM.      *;
54    %PUT ************************************************************;
55    %PUT # OF OBS IN SASHELP.CARS           = %CMPRES(&COUNT_CARS);
56    %PUT # OF OBS IN WORK.CARS              = %CMPRES(&COUNT_SAVINGS);
57    %PUT ************************************************************;
58 %MEND LOG_REPORT;
```

SAMPLE LOG REPORT

When the program with the misspelled variable and the misspelled data set is run, the Log looks like this:

```
141         DATA CARS;
142             SET SASHELP.CARS;
143             IF ORIGIN EQ 'USA';
144             DISCOUNT = MSRP - INVOIC;
145         RUN;

NOTE: Variable INVOIC 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).
      147 at 144:21
NOTE: There were 428 observations read from the data set
SASHELP.CARS.
NOTE: The data set WORK.CARS has 147 observations and 17 variables.

146
147         PROC PRINT DATA = CAR;
ERROR: File WORK.CAR.DATA does not exist.
148         RUN;

NOTE: The SAS System stopped processing this step because of errors.
NOTE: PROCEDURE PRINT used (Total process time):
```

Log 1. Portion of Log show program with problems

You can visually spot syntax ERRORs and run time ERRORs because they appear in red. It is trickier to find other kinds of errors such as uninitialized variables and generated missing values as they appear as green NOTEs and can blend in the rest of the NOTEs.

The Log Report generated by the LOG_REPORT macro helps identify non-syntax ERRORs and non-runtime ERRORS as it looks for problems listed in the NOTEs.

The Log Report for program with misspellings looks like this :

```
************************************************************
* GENERAL SAS ALERTS:                                      *
************************************************************
*   NORMALLY, BELOW ALERTS SHOULD BE ZERO                  *
*   IF THEY DO NOT HAVE ZERO INSTANCES                     *
*   DETERMINE IF THERE IS AN ISSUE.                        *
************************************************************
# OF ERRORS                       = 1
# OF WARNINGS                     = 0
# OF UNINITIALIZED VARIABLES      = 1
# OF MISSING VALUES               = 1
************************************************************
*   THIS SECTION SHOWS THE FLOW OF DATA IN THE PROGRAM.    *
************************************************************
# OF OBS IN SASHELP.CARS          = 428
# OF OBS IN WORK.CARS             = 147
************************************************************
```

Log 2. Portion of Log showing Log Report

CONCLUSION

There are many ways to ensure that programs run properly. A Log Report is one tool that can easily identify instances of terms that can be indicative of problems. You can adapt the LOG_REPORT macro to provide additional information about the program run and the flow of data.

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

REFERENCES

Knapp, Peter. 2003. “Debugging 101” *Proceedings of the SAS Users Group International 2004 Conference*, Montréal, Canada. Available at https://support.sas.com/resources/papers/proceedings/proceedings/sugi29/257-29.pdf.

ACKNOWLEDGMENTS

I would like to thank my colleague Girish Narayandas who adapted and improved the Log Report macro I originally wrote 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