SAS® GLOBAL FORUM 2021

PAPER 1049-2021

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. The Log Report was developed 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. 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 run along with information about the data flow.

REVIEWING THE LOG FOR PROGRAM RUN PROBLEMS

One of the golden rules when running SAS programs is to review the Log for problems. Using the Log Summary built into SAS Enterprise Guide and SAS Studio it's easy to find Syntax Errors and Warnings.

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

- Uninitialized variables
- Missing values
- Repeats of by values

- Converted variables
- Division by zero detected
- Invalid data values
- Rejected missing weight variable values

SAMPLE PROGRAM

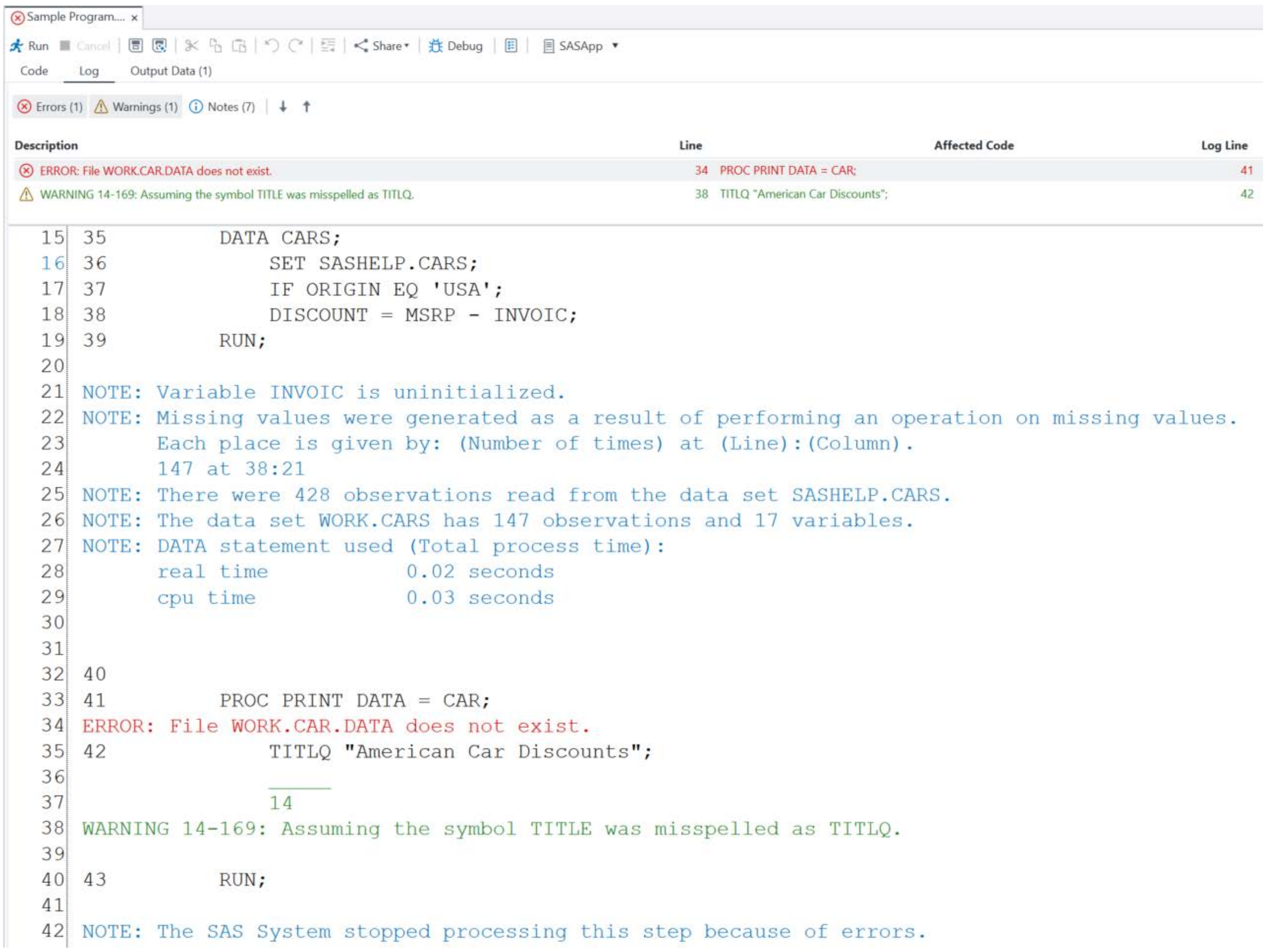
When the following example program is run it generates an uninitialized variable NOTE, a run time ERROR, and WARNING message:

```
DATA CARS;
    SET SASHELP.CARS;
    IF ORIGIN EQ 'USA';
    DISCOUNT = MSRP - INVOIC;
RUN;

PROC PRINT DATA = CAR;
    TITLQ "American Car Discounts";
RUN;
```

SAS ENTERPRISE GUIDE LOG

When running the program in SAS Enterprise Guide the log looks like this:



Description	Line	Affected Code	Log Line
ERROR: File WORK.CAR.DATA does not exist.	34	PROC PRINT DATA = CAR;	41
WARNING 14-169: Assuming the symbol TITLE was misspelled as TITLQ.	38	TITLQ "American Car Discounts";	42

```
35          DATA CARS;
36              SET SASHELP.CARS;
37              IF ORIGIN EQ 'USA';
38              DISCOUNT = MSRP - INVOIC;
39          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 38:21
NOTE: There were 428 observations read from the data set SASHELP.CARS.
NOTE: The data set WORK.CARS has 147 observations and 17 variables.
NOTE: DATA statement used (Total process time):
      real time           0.02 seconds
      cpu time            0.03 seconds


40
41          PROC PRINT DATA = CAR;
ERROR: File WORK.CAR.DATA does not exist.
42              TITLQ "American Car Discounts";
                _____
                14
WARNING 14-169: Assuming the symbol TITLE was misspelled as TITLQ.

43          RUN;

NOTE: The SAS System stopped processing this step because of errors.
```

Sample Log in SAS Enterprise Guide

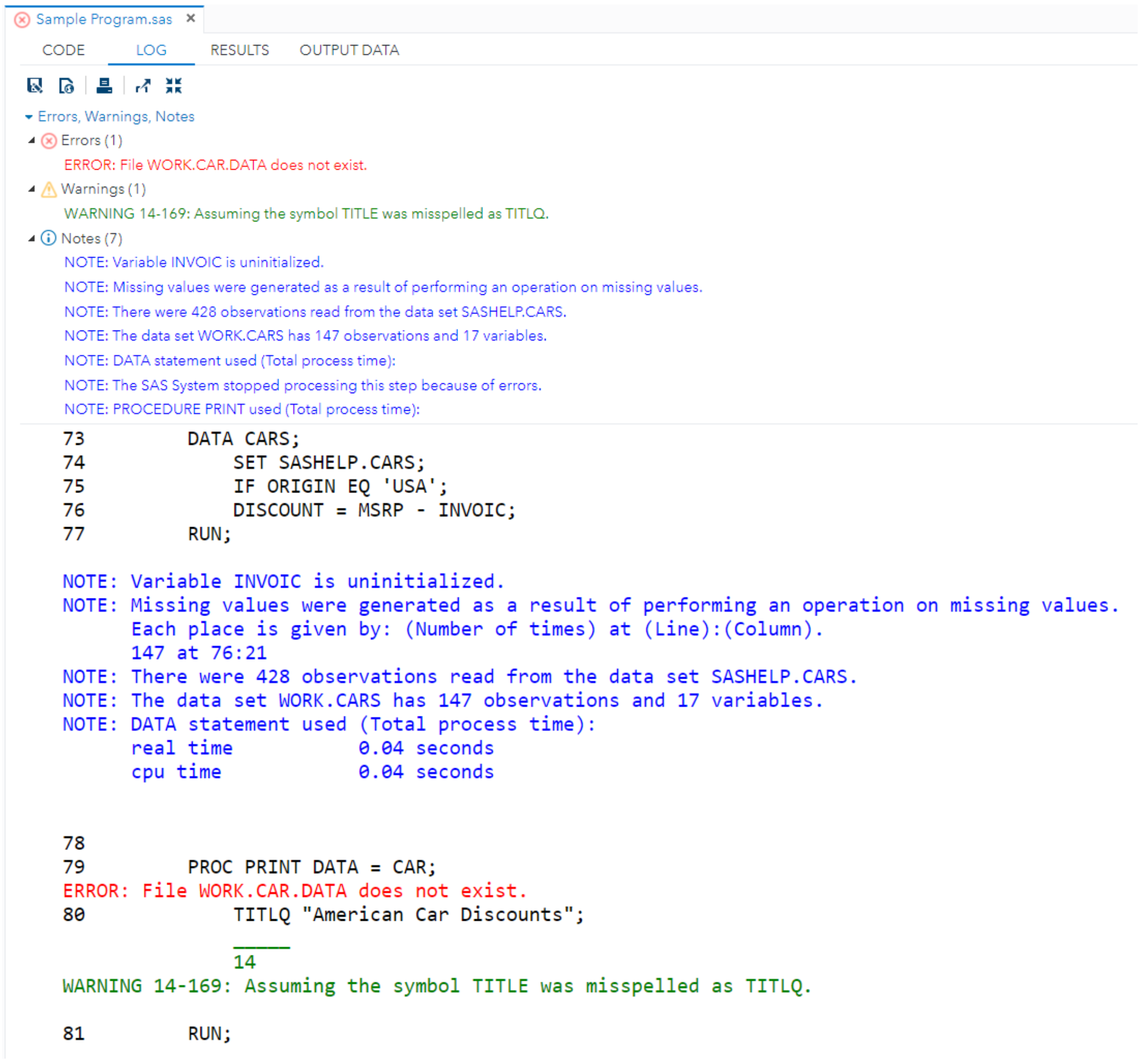
In the Log Summary the Errors tab indicates the total number of syntax and run time errors. The Description section lists the error messages and the ERROR messages appear in the Log. Clicking on an ERROR message in the Description section will highlight the error listed in the Log.

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.

SAS STUDIO LOG

When running the program in SAS Studio the log looks like this:



```
Errors, Warnings, Notes
  Errors (1)
     ERROR: File WORK.CAR.DATA does not exist.
  Warnings (1)
     WARNING 14-169: Assuming the symbol TITLE was misspelled as TITLQ.
  Notes (7)
     NOTE: Variable INVOIC is uninitialized.
     NOTE: Missing values were generated as a result of performing an operation on missing values.
     NOTE: There were 428 observations read from the data set SASHELP.CARS.
     NOTE: The data set WORK.CARS has 147 observations and 17 variables.
     NOTE: DATA statement used (Total process time):
     NOTE: The SAS System stopped processing this step because of errors.
     NOTE: PROCEDURE PRINT used (Total process time):
```

```
73          DATA CARS;
74              SET SASHELP.CARS;
75              IF ORIGIN EQ 'USA';
76              DISCOUNT = MSRP - INVOIC;
77          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 76:21
NOTE: There were 428 observations read from the data set SASHELP.CARS.
NOTE: The data set WORK.CARS has 147 observations and 17 variables.
NOTE: DATA statement used (Total process time):
      real time           0.04 seconds
      cpu time            0.04 seconds


78
79          PROC PRINT DATA = CAR;
ERROR: File WORK.CAR.DATA does not exist.
80              TITLQ "American Car Discounts";
                _____
                14
WARNING 14-169: Assuming the symbol TITLE was misspelled as TITLQ.

81          RUN;
```

Sample Log in SAS Studio

In the Log Summary the total number of errors are displayed in the Errors section along with a listing of the errors. Clicking on an ERROR message in the Errors section will highlight the error listed in the Log. The Warning section behaves the same way as does the Notes section.

As with SAS Enterprise Guide the SAS Studio 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:

- Redirect the Log to an external file.
- Run the program to be reviewed.
- Reset the Log Destination to the Log.
- Run the Log Report macro.

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 = FILEREF NEW;
RUN;
```

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 CARS;
    SET SASHELP.CARS;
    IF ORIGIN EQ 'USA';
    DISCOUNT = MSRP - INVOIC;
RUN;

PROC PRINT DATA = CAR;
    TITLQ "CAR DISCOUNTS";
RUN;
```

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;
```

RUN THE LOG REPORT MACRO

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

- Copies the saved Log to the default log destination.
- Reads the saved Log.
- Looks for key words.
- Accumulates instances of key words.
- Saves key word totals into macro variables.
- Writes key word totals to the Log Report.

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.

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.

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.

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;
```

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;
    %PUT *************************************************************;
    %PUT * PROGRAM FLOW OF DATA IN THE PROGRAM: Obs before and after. *;
    %PUT *************************************************************;
    %PUT # OF CARS IN SASHELP.CARS  = %CMPRES(&COUNT_CARS);
    %PUT # OF USA CARS IN WORK.CARS = %CMPRES(&COUNT_SAVINGS);
%MEND LOG_REPORT;
```

RESULTS OF LOG REPORT MACRO RUN

The following Log shows the Log Report macro run which does the following six things:

- Copies the saved Log to the default log destination.
- Reads the saved Log.
- Routes the log to the default log destination.
- Looks for key words.
- Saves key word totals into macro variables.
- Writes key word totals to the Log Report using %PUT statements.

Note: %PUT statements don't appear in the Log:

```
106         %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=C:\Sample Program.log,
      RECFM=V,LRECL=32767,File Size (bytes)=1357,
      Last Modified=10May2021:21:16:13,
      Create Time=01Mar2021:22:30:03

92
93          DATA CARS;
94              SET SASHELP.CARS;
95              IF ORIGIN EQ 'USA';
96              DISCOUNT = MSRP - INVOIC;
97          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 96:21
NOTE: There were 428 observations read from the data set SASHELP.CARS.
NOTE: The data set WORK.CARS has 147 observations and 17 variables.

98
99          PROC PRINT DATA = CAR;
ERROR: File WORK.CAR.DATA does not exist.
```

```
100             TITLQ "CAR DISCOUNTS";
                _____
                14
WARNING 14-169: Assuming the symbol TITLE was misspelled as TITLQ.

101         RUN;

NOTE: The SAS System stopped processing this step because of errors.

102
103         PROC PRINTTO LOG = LOG;
104         RUN;
NOTE: 44 records were read from the infile FILEREF.
      The minimum record length was 0.
      The maximum record length was 93.

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,1,6))) = "ERROR:" THEN
ERROR + 1;
MPRINT(LOG_REPORT):   ELSE IF UPCASE(COMPRESS(SUBSTR(LINE,1,8))) = "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= C:\Sample Program.log,
      RECFM=V,LRECL=32767,File Size (bytes)=1357,
      Last Modified=10May2021:21:16:13,
      Create Time=01Mar2021:22:30:03

NOTE: 44 records were read from the infile FILEREF.
      The minimum record length was 0.
      The maximum record length was 93.

**************************************************************
* GENERAL SAS ALERTS: Determine cause of non-zero instances. *
**************************************************************
# OF ERRORS                  = 1
# OF WARNINGS                = 0
# 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 CARS dataset from the SASHELP library and keeps cars originating in the USA:

```
DATA CARS;
    SET SASHELP.CARS;
    IF ORIGIN EQ 'USA';
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.CARS;
    QUIT;

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

    %PUT **************************************************************;
    %PUT * PROGRAM FLOW OF DATA IN THE PROGRAM: Obs before and after. *;
    %PUT **************************************************************;
    %PUT # OF CARS IN SASHELP.CARS  = %CMPRES(&COUNT_CARS);
    %PUT # OF USA CARS IN WORK.CARS = %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.CARS;
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.CARS;
MPRINT(LOG_REPORT):   QUIT;
NOTE: PROCEDURE SQL used (Total process time):
      real time           0.00 seconds
      cpu time            0.01 seconds
```

```
**************************************************************
* PROGRAM FLOW OF DATA IN THE PROGRAM: Obs before and after. *
**************************************************************
# OF CARS IN SASHELP.CARS  = 428
# OF USA CARS IN WORK.CARS = 147
```

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. 2020. “Log Reviewing Made Easy.” *Proceedings of the Southeast SAS User Group 2020 Conference*, Savannah, Georgia. Available at https://sesug.org/proceedings/sesug_2020_final_papers/Know_Your_SAS__Advanced_Techniques/SESUG2020_Paper_112_Final_PDF.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