SAS Users Group Crash Course: Log Parsing

How do you prove your SAS program did what it was supposed to do?

Contents

[Find the SAS Users Group on Slack: #sas-users-group 1](#_Toc36023809)

[The basics 1](#_Toc36023810)

[Good SAS .log practices 2](#_Toc36023811)

[PROC PRINTTO and Batch Submit 2](#_Toc36023812)

[Check ERROR: and WARNING: messages 2](#_Toc36023813)

[Using a PUT statement to add ERROR: and WARNING: to the log 2](#_Toc36023814)

[Resolve macro variables 3](#_Toc36023815)

[What else do you need to check? 4](#_Toc36023816)

[Logparse macro 4](#_Toc36023817)

[Scanlogs macro 5](#_Toc36023818)

# Find the SAS Users Group on Slack: #sas-users-group

For any and all SAS code questions and cool tricks, as well as SAS-related announcements.

# The basics

SAS logs are produced every time you run SAS code and provide a record of everything that happened in that code. The log file includes:

* the names of the data sets that you create,
* the number of observations and variables in those data sets,
* what statements were executed,
* how much time the DATA and PROC steps required, and
* warnings and error messages.

At a minimum, you need to check your log every time you run SAS code to ensure it ran as expected. ‘Running as expected’ includes code that runs correctly (without errors) and produce the output you planned. This session is covers some strategies and tips to make your SAS log-checking more informative and efficient.

Log checks can be done in the SAS programming environment or you can save the log into a separate file using PROC PRINTTO. In SAS Studio and SAS Enterprise Guide, the log generates in a separate tab. You can select a warning/error/note to be taken to the location of that message in the log. If you are coding in the standard desktop version of SAS, you have to manually scroll through to find these messages. SAS color codes errors as red, warnings are green, and notes as blue.

# Good SAS .log practices

## PROC PRINTTO and Batch Submit

It is good practice to maintain a record of your log files when you are running “finalized” programs. This can be done programmatically with PROC PRINTTO:

**proc** **printto** print="Documents\logparse\_listing.lst"

log="Documents\logparse.log" NEW;

**run**;

**proc** **freq** data=sashelp.cars;

table Cylinders;

**run**;

**proc** **printto**;**run**;

This code will send your .log and .lst output to whatever directory you specify. These can be examined like any text file, by manually scrolling or until CTRL-F to look for keywords.

You can also create a record for your .log and .lst files by right clicking on a SAS program and selecting ‘Batch submit with SAS [whatever version]’. The .log and .lst files will be created in the same directory as the SAS program.

## Check ERROR: and WARNING: messages

The two things that I learned to check for my in SAS logs in my Base SAS class was WARNING: and ERROR:. While WARNING: messages may be permissible in a properly-compiled program, ERROR: messages are never okay to leave unresolved. WARNING: messages should always be investigated to ensure the warning is expected. If you are not going to resolve the warning message, I would suggest including a comment at the top of program (where your program summary comment is) indicating which warning messages are expected and why so future programmers (including you) know.

**proc** **print** data=sashelp.cars;

var mpg Cylinders;

**run**;

11 proc print data=sashelp.cars;

12 var mpg Cylinders;

ERROR: Variable MPG not found.

13 run;

dat cars;

set sashelp.cars;

**run**;

36 dat cars;

---

14

WARNING 14-169: Assuming the symbol DATA was misspelled as dat.

37 set sashelp.cars;

38 run;

## Using a PUT statement to add ERROR: and WARNING: to the log

SAS will help you out by generating ERROR and WARNING messages for routine deviations, you can use a PUT statement to generate more specific ERROR and WARNING messages. Using PUT statements in the way can help you ascertain if code is producing accurate output, even when it would compile without automatically generated ERROR messages. As an example, you can use a PUT statement to alert you if you are reading in different data tables and know that certain conditions may arise – eg, null entries – that will cause programs for your output that may not generate routine SAS ERROR or WARNING messages.

**data** shoes;

set sashelp.shoes;

by region subsidiary product;

if first.product and not last.product then put 'WARNING: Multiple products in ' subsidiary / product=;

**run;**

39 data shoes;

40 set sashelp.shoes;

41 by region subsidiary product;

42 if first.product and not last.product then put 'WARNING: Multiple products in '

42 ! subsidiary / product=;

43 run;

WARNING: Multiple products in Copenhagen

Product=Sport Shoe

NOTE: There were 395 observations read from the data set SASHELP.SHOES.

NOTE: The data set WORK.SHOES has 395 observations and 7 variables.

NOTE: DATA statement used (Total process time):

real time 0.20 seconds

cpu time 0.11 seconds

## Resolve macro variables

If you love macro variables like me, it is imperative to ensure that these are resolving to appropriate values within your code – it is relatively easy to get slopping with macro variables in a way that will not generate problems until you start analyzing your output.

The SYMBOLGEN option can be used to verify how macro variables resolve throughout the program.

options symbolgen;

%let high\_MSRP=85000;

**proc** **sql**;

create table high\_MSRP as

select \*

from sashelp.cars

where MSRP >= &high\_MSRP;

**quit**;

34 options symbolgen;

35 %let high\_MSRP=85000;

36

37 proc sql;

38 create table high\_MSRP as

39 select \*

40 from sashelp.cars

41 where MSRP >= &high\_MSRP;

SYMBOLGEN: Macro variable HIGH\_MSRP resolves to 85000

NOTE: Table WORK.HIGH\_MSRP created, with 9 rows and 15 columns.

42 quit;

Alternatively, %put statements within your code to resolve macro variables in your log a single time.

%let high\_MSRP=75000;

%put &high\_MSRP;

9 %let high\_MSRP=75000;

10 %put &high\_MSRP;

75000

Other options that can be used to resolve macro variables and functions are MLOGIC and MPRINT. These can make logs very long and are more typically used for debugging.

## What else do you need to check?

When developing a SAS program, it is good practice to establish a plan to check over your log and lst files in addition to your output. Some common checks to consider include:

* Distinct observation, null, and total counts – have duplicates been inadvertently introduced? Are there more NULL values than expected?
* Observation counts post-subsetting or join – do the counts make sense?
* Frequencies for recoded variables vs. the original variable – did the recoding produce the expected categories?
* PROC MEANS or PROC UNIVARIATE of numeric values in final data tables – are minimums, maximums, standard deviations, etc. sensible?

# Logparse macro

The logparse macro is can be used to identify the time it took to run DATA and PROC steps in your code. This can be used to identify overly time-consuming steps that may be more efficiently conducted with a different procedure or strategy. The Excel file included in the course materials shows an example of what the output from this macro looks like.

The output from the logparse macro contains a row for every PROC or DATA step in your log and columns contain information about that PROC or DATA step. The timing columns realtime and cputime are going to help identify steps that are slowing down your code that you may be able to strategize to run faster. This information is also very helpful if you are benchmarking different strategies to find out what will run faster.

This is the paper that the logparse macro is published:

<https://support.sas.com/resources/papers/proceedings/proceedings/sugi30/219-30.pdf>

This is the link for the Urban GitHub repo with the macro:

<https://github.com/UI-Research/Logparse>

# Scanlogs macro

The scanlogs macro can be used to quickly identify specific keywords for all logs in a given directory. The basic code includes ERROR: and WARNING: messages, but this can be customized to include any word or phrase that is pertinent to your code. The html file included in the course materials shows an example of what the output from this macro looks like.

The output from the scanlogs shows each .log file in a directory and how many times an error, warning, or other text string from the macro occurs in the file. Note that this macro uses X commands, which is how you can use command prompt code within SAS, and this feature is automatically turned off in SAS Studio.

This is the paper that the scanlogs macro is published:

<https://www.lexjansen.com/pharmasug/2009/po/PO25.pdf>