PharmaSUG 2026 Paper AP-011

Take CoMmanD of Your Log:
Using CMD to Check Your SAS® Program Logs

Richann Jean Watson, DataRich Consulting

ABSTRACT

Regardless of the industry, part of writing a SAS® program is to ensure that the log is free of any unwanted log messages. When running the program in an interactive SAS session, we can review the log as we execute the program and SAS is good about highlighting ERROR and WARNING messages using colors to draw the eye. Other types of unwanted log messages, such as INFO, uninitialized, character to numeric conversion, may not be so easily spotted. When running the program in batch, each program needs to be opened and scanned for unwanted log message, which is tedious and prone to overlooking a message. There have been several papers illustrating the creation of macros that will check the logs by parsing the logs after the programs have been executed. While these macros are great when you are running a lot of programs for a deliverable and need to check all the logs, these check log macros are not necessarily ideal during development. It is during development that we need to ensure the program is running clean. Although we could possibly use the same macro that is used to check all the programs and filter to run on one program, that would require us to run an extra program. What if there is an easier way? This paper demonstrates the use of the command line interface to execute the program in batch as well as check the log and provide a summary.

INTRODUCTION

After we create our programming masterpiece (admit it some SAS programs are works of art!), we need to make sure that the program is going to run clean. In other words, we have to make sure there are no unwanted log messages. Some industries may look for the dreaded ERROR and WARNING and that may be sufficient for their needs, but some industries may have stricter rules and want all unwanted log messages removed from the program. When running each program interactively, we can check for these unwanted log messages and clean the program up. If the program is run in batch to ensure it will run successfully and clean in batch mode, we need to open the log and examine it for those pesky unwanted log messages.

Note that the focus of this paper is reviewing the logs while a program is in development and not necessarily while a program is already in production and part of a production run. Programs that are part of a production run are typically run in one large batch file or via a driver program and a macro will check all the logs for all the programs run. If you would like details on a macro that checks multiple programs logs and produces a report, refer to “Check Please: An Automated Approach to Log Checking” (Watson, Check Please: An Automated Approach to Log Checking, 2017).

THE FULL PROCESS

To understand the different steps in the script, an overview of the entire process is needed. The script determines where the log files are saved. It then kicks off the execution of the program. When the program begins execution, the date and time of initiation is preserved. After the execution of the program, the date and time of completion is preserved. This is important because the original log generated by the program will have summaries prepended, thus overwriting the original log date/time stamp. Once the program completes execution, then the log generated by the program is scanned for unwanted log messages. Each unwanted log message is then written to a temporary file. A secondary temporary file is also created that counts the number of times a particular unwanted log message was found in the original log. These two summaries are then prepended to the original log; allowing the user to open the log and quickly identify if the program executed by examining the date/time stamps and identify if there were any unwanted log messages by examining the summaries.

TYPES OF UNWANTED LOG MESSAGES

There are a variety of log messages that we may encounter during the development of our program. How these messages are dealt with may be dependent on the data (e.g., is it messy data and it stays as an unwanted message).

Some of the more common messages contain the following word or phrase:

- ERROR
- WARNING
- UNINITIALIZED
- MERGE statement has more than
- SAS SYSTEM STOPPED PROCESSING
- Values have been converted
- MISSING VALUES WERE GENERATE
- INVALID DATA
- INVALID NUMERIC DATA
- W.D. format to small
- ONE OR MORE LINES WERE TRUNCATED
- variables have conflicting

Additional log messages that may be encountered include:

- FATAL
- Unequal
- ILLEGAL
- SAS went to a new line when
- DATA STEP STOPPED
- STATEMENT NOT EXECUTED
- Invalid argument
- NOTE: LOST CARD
- DOES NOT EXIST
- COULD NOT BE LOADED
- Division by zero detected
- Operations could not be performed
- Interactivity disabled with
- duplicate key values were deleted
- INFO: The variable
- WHERE CLAUSE HAD BEEN REPLACED

CHECKING LOGS

At the start of the project, it should be decided what kind of log messages are acceptable. For example, user-defined log messages may be acceptable if they are triggered by discrepancies or issues in the data.

PROBLEM WITH MANUAL CHECKS

While running a program interactively, we can run each step individually and examine the log for each of the unwanted messages. This approach is great during the initial development of the program. However, to ensure the program is running successfully, we would either need to start a fresh SAS session and run the program from beginning to end or run the program using the right-click ‘Batch Submit with SAS 9.4’. This prevents left-over code, data sets, or macro variables from masking problems in the program. Either technique now requires us to look through the log for each of these unwanted log messages. For short programs this would be easy but for programs that are long and complex (e.g., code that is 500+ lines or code that has several macro calls or code that uses techniques such as CALL EXECUTE), this can be overwhelming and may cause occasional headaches. We may be tempted to say, ‘it ran successfully when I developed it, it should still run successfully in batch.’ Unfortunately, that is not always the case.

CMD TO THE RESCUE

This paper introduces an approach to log checking during program development using the command line interface (formerly known as CLI, currently known as CMD). The CMD script will execute the program and then scan the log for various unwanted log messages. In addition, it creates two types of summaries. One summary is the number of each type of unwanted log message. The second summary is a compiled list of these unwanted messages. Both summaries are then prepended to the original log for a quick review.

This paper will not go into detail regarding what each of the various commands represent. For details on several of the commands refer to “Going Command(o): Power(Shell)ing Through Your Workload” (Watson & Hadden, 2023) or for a full list of Windows CMD commands, options and extensive details visit SS64.com (Sheppard, An A-Z Index of Windows CMD commands., n.d.).

Initialize Key Components

Within CMD there are parameters which are arguments that are passed in a batch script (Sheppard, How-to: Pass Command Line arguments (Parameters) to a Windows batch file, n.d.) and environment variables which are set during the session and only available during the session (Sheppard, How-to: Windows Environment Variables, n.d.). Refer to CMD Basics - Variables Versus Parameters in "Going Command(o): Power(Shell)ing Through Your Workload" (Watson & Hadden, 2023) for illustration between the parameters and environment variables.

Parameter

Parameter values are retrieved by using its position on the command line preceded by a %. For example, %1 indicates the parameter value for the argument in the first position should be retrieved. The use of %* indicates that it applies to all numerical positions (e.g., %1 %2 %3 … %99 … etc.) and only arguments in positions %1 through %9 can be referenced by number. In a FOR loop, a parameter is preceded with %%. (Sheppard, How-to: Pass Command Line arguments (Parameters) to a Windows batch file, n.d.).

One of the first steps in the CMD script is to set an environment variable, **pgm**, that will be used throughout the remainder of the script. The environment variable is based on an argument (i.e., parameter, **%***) that is passed into the script. The argument is then assigned to the parameter %%A and all commands within the DO block are performed on the parameter %%A. In this case we are assigning the environment variable, **pgm**. %%~nA is used to extract the name of the file without adding the extension (Sheppard, How-to: Pass Command Line arguments (Parameters) to a Windows batch file, n.d.).

```
for %%A in (%*) do set pgm=%%~nA
```

For example, if we are executing program with the name of "CMD-Checklog-Test.sas", then the environment variable, **pgm**, would be set to "CMD-Checklog-Test". Notice it does not have the ".sas" extension.

Environment Variables

Environment variables are different than parameters in that an environment variable is assigned during the script where a parameter is passed to script. Environment variables are only available during the current script.

Once the **pgm** environment variable is set based on the parameter, %%A, that is passed to the script. The next part of the script initializes the execution start time, location of LOG and LST files and opens a temporary file.

- Execution start time is based on the current time that the batch script starts and is used to calculate elapsed time for execution. The SET command is used to create an environment variable, **sttime**, which will be used later in the script.

  ```
  set sttime=%time%
  ```

- The script looks to see if there are subfolders (Logs, Lsts) under the directory where the program resides and if so, assigns that as the location where the LOG and LST files should be written. If no subdirectory exists, then it checks to see if there are folders at the same level as the program folder and if so, assigns that as the location where the LOG and LST files should be written. If no such directory exists, then it will default to writing the LOG and LST files to the same location as the program. The location for the LOG files is defined with the environment variable, **LogFldr**, and the location for the LST files is defined with the environment variable, **LstFldr**.

  ```
  if exist "%~d1%~p1Logs" (set LogFldr=%~d1%~p1Logs) else if exist
  "%~d1%~p1..\Logs" (set LogFldr=%~d1%~p1..\Logs) else (set
  LogFldr=%~d1%~p1)
  ```

  ```
  if exist "%~d1%~p1Lsts" (set LstFldr=%~d1%~p1Lsts) else if exist
  "%~d1%~p1..\Lsts" (set LstFldr=%~d1%~p1..\Lsts) else (set
  LstFldr=%~d1%~p1)
  ```

- A temporary file named "__logck_summary_%pgm%.txt" is initialized with the start time, and stored in the location of the LOG and LST files. The temporary file will contain the summary information about execution and counts of unwanted log messages. Note that for the temporary file, %pgm% is determined by the name of the program that is being executed. This prevents a 'crash' between two programmers trying to write in the same file. The single > indicates that the file is to be overwritten, while >> indicates the file is to be appended to.

```
set LogCk=%LogFldr%
ECHO Batch start at %sttime%   > "%LogCk%\__logck_summary_%pgm%.txt"
ECHO LOG=%LogFldr%         >> "%LogCk%\__logck_summary_%pgm%.txt"
ECHO LST=%LstFldr%         >> "%LogCk%\__logck_summary_%pgm%.txt"
```

Note an environment variable is only available during the current session and can be referenced by enclosing it in % (Sheppard, How-to: Windows Environment Variables, n.d.).

Submit SAS Program

The next part is submitting the SAS program. To do so we need to set the environment variable that will call the the SAS executable (sas). In addition, we need to set the environment variables associated with writing the LOG (plog) and LST (plst) files during the batch submit.

After we set the environment variables, we write to the temporary file the name of the program. This is not necessary, but it is a nice to confirm in the report that we are looking at the correct program.

```
set sas="C:\Program Files\SASHome\SASFoundation\9.4\sas.exe" -CONFIG
"C:\Program Files\SASHome\SASFoundation\9.4\nls\en\sasv9.cfg"
set plog=-log "%LogFldr%"
set plst=-print "%LstFldr%"
ECHO Program:   %pgm%   >> "%LogCk%\__logck_summary_%pgm%.txt"
```

Using environment variables, we can then start the batch submit. In the CMD command we reference the sas, plog and plst environment variables. In addition, for this particular example, we specify the autoexec and the location where the autoexec resides (%~d1%~p1). In this case ~d1 represents the drive letter and ~p1 represents the path where the program resides. Note that if there is no autoexec, this can be removed.

```
%sas% -autoexec "%~d1%~p1autoexec.sas"-sysin "%~d1%~p1%pgm%.sas" %plog%
%plst% -icon -nosplash -rsasuser
```

The time of completion of the batch submit is then written to the temporary file.

```
ECHO '%pgm%' Completed: -!date! !time! >> "%LogCk%\__logck_summary_%pgm%.txt"
```

Check the Log

Now that the SAS program has been executed, we have to check the log to make sure it is executed successfully.

To parse the log for various unwanted log messages, we are going to use the FINDSTR command. The FINDSTR command searches for a string in a file or multiple files. Although FINDSTR allows the use of complex regular expressions, we do not use regular expressions in this script. (Sheppard, FINDSTR, n.d.)

FINDSTR	Syntax	Description	
Searches for a string in a file(s) Allows for use of complex regular expressions	FINDSTR *string(s)* [*pathname(s)*] [/C:"*string*"] [/OFF[LINE]] [options]	*string(s)*	text to search for if there are mutliple words, then each word is a separate search
		Optional Keys:	
		pathname(s)	file(s) to search
		/C: *string*	use the string as a literal search (may include spaces)
		Optional Options:	
		/OFF[LINE]	do not skip files with offline attributes
		/I	case-insensitive search
		/N	print the line number before each line match

For a complete list of keys and options available for FINDSTR, visit ss64.com (Sheppard op. cit).

Table 1 illustrates the use of FINDSTR and several of the key arguments and options. If the FINDSTR is successful, then the line is being directed to the external file, "__logck_%pgm%.txt". Note that the temporary file "__logck_%pgm%.txt" was initialized prior to the scanning (not shown). Therefore, we are going to append to the existing file using >> so as not to overwrite the existing file. Note that this file is different than the temporary file that will contain the summary information ("__logck_summary_%pgm%.txt"). In Table 1 not all commands are listed, only a sample is provided. For a complete list of possible log messages being searched for refer to the program found on https://github.com/rwatson724/CMD-CheckLog.

COMMAND	DESCRIPTION
`findstr /n /OFFLINE "ERROR:" "%LogFldr%\%pgm%.log" >> "%LogCk%\__logck_%pgm%.txt"`	Looks for the word "ERROR" followed by a colon in the indicated file and is case sensitive.
`findstr /n /OFFLINE /i "FATAL" "%LogFldr%\%pgm%.log" >> "%LogCk%\__logck_%pgm%.txt"`	Looks for the word "FATAL" in the indicated file and it is case-insensitive (/i).
`findstr /n /OFFLINE "WARNING" "%LogFldr%\%pgm%.log" >> "%LogCk%\__logck_%pgm%.txt"`	Looks for the word "WARNING" in the indicated file
`findstr /n /OFFLINE /i "uninitialized unequal ILLEGAL" "%LogFldr%\%pgm%.log" >> "%LogCk%\__logck_%pgm%.txt"`	Looks for the word "uninitialized", "unequal" or "illegal" in the indicated file. Each word is searched separately, and it is case-insensitive (/i).
`findstr /n /OFFLINE /i /c:"SAS went to a new line when" "%LogFldr%\%pgm%.log" >> "%LogCk%\__logck_%pgm%.txt"`	Looks for the **exact** phrase (/c:) "SAS went to a new line when" in the indicated file and it is case-insensitive (/i).

Table 1: Samples of FINDSTR in the Check Log Script

Produce the Summary Report

After identifying all the unwanted log messages, we want to produce some sort of summary report. The report creation is done in phases.

Count of Log Message Type

In addition to searching for the log messages, a counter can also be set up to count the number of occurrences of the different types of log messages.

To count the number of each type of log messages, we first initialize a counter using the SET command. SET allows us to create an environment variable and assign a value (Sheppard, SET, n.d.). After we initialized the environment variables that will be used as counters, we then used the FINDSTR within the FOR command to count the number of occurrences. Similar to a DO loop, FOR executes a command each time a condition (i.e., FINDSTR) is met (Watson & Hadden, Going Command(o): Power(Shell)ing Through Your Workload, 2023; Sheppard, FOR, n.d.).

SET	Syntax	Description	
Create, display or remove an environment variable, which is only available during the current CMD. The variable name itself is not case sensitive but the value can be and spacing is NOT ignored.	SET *variable* SET *variable=string* SET "*variable=string*" SET "*variable* =" SET /A "*variable=expression* " SET /P *variable=[promptString*] SET "	*variable* *string* *expression* /A /P	new or existing environment variable name text string to assign to the variable arithmetic expression see documentation for details on arithmetic expressions prompt for user input To delete a variable use SET "variable=". This will ensure there is no trailing space after the equal sign. You can also use (SET variable=).

For more details on SET, visit ss64.com (Sheppard op. cit).

FOR	Syntax	Description	
Perform a *command* multiple times if the condition is met. A *command* that contains complex logic (i.e., multiple commands) is embedded in parenthesis. Note if running in a batch program, then %% precedes the *parameter*. If running at the command line, then % precedes the *parameter*.	FOR /F ["options"] %%*parameter* IN (*filenameset*) DO *command* FOR /F ["options"] %%*parameter* IN ("*Text string to process* ") DO *command*	%%parameter is set to a value for each iteration of the FOR loop that is executed on a set of files. *Filenameset* is a one or more files. For a *filenameset* it parses the contents of the file one line at at time. The contents in the file are broken up into tokens. **Available Options:**	delims = xxx delimiter character(s) default for strings is a space or TAB tokens = n indicates which numbered items to read from each line default is 1 multiple tokens are separated by a comma

For a complete list of keys and options available for FOR, visit ss64.com (Sheppard op.cit).

Table 2 illustrates the use of SET and FOR (with FINDSTR) to count the number of occurrences of a particular log message. If the FINDSTR within the FOR command is successful, then the environment variable is incremented.

COMMAND	DESCRIPTION
`set /A TtlNumErr=0` `for /f %%j in ('findstr "ERROR" "%LogFldr%\%pgm%.log"') do @set /A TtlNumErr+=1`	Initialize (SET) the environment variable ***TtlNumErr*** to 0. Loop through each line in the log looking for the word "ERROR" (case sensitive) and if found increment the environment variable by 1.
`set /A TtlNumWrn=0` `for /f %%j in ('findstr "WARNING" "%LogFldr%\%pgm%.log"') do @set /A TtlNumWrn+=1`	Initialize (SET) the environment variable ***TtlNumWrn*** to 0. Loop through each line in the log looking for the word "WARNING" (case sensitive) and if found increment the environment variable by 1.
`set /A TtlNumInf=0` `for /f %%j in ('findstr /i /c: "INFO: The variable" "%LogFldr%\%pgm%.log"') do @set /A TtlNumInf+=1`	Initialize (SET) the environment variable ***TtlNumInf*** to 0. Loop through each line in the log looking for the **exact** phrase (/c:) "INFO: The variable" which is case-insensitive (/i) and if found increment the environment variable by 1.

COMMAND	DESCRIPTION
`set /A TtlNumMsc=0` `for /f %%j in ('findstr /i "FATAL" "%LogFldr%\%pgm%.log"') do @set /A TtlNumMsc+=1`	Initialize (SET) the environment variable ***TtlNumMsc*** to 0. Loop through each line in the log looking for the word "FATAL" which is case-insensitive (**/i**) and if found increment the environment variable by 1.
`for /f %%j in ('findstr /i "unequal ILLEGAL" "%LogFldr%\%pgm%.log"') do @set /A TtlNumMsc+=1`	Loop through each line in the log looking for the word "unequal" or "ILLEGAL" which is case-insensitive (**/i**) and if found increment the environment variable by 1.
`for /f %%j in ('findstr /i /c:"THE SAS SYSTEM STOPPED PROCESSING" "%LogFldr%\%pgm%.log"') do @set /A TtlNumMsc+=1`	Loop through each line in the log looking for the **exact** phrase (**/c:**) "THE SAS SYSTEM STOPPED PROCESSING" which is case-insensitive (**/i**) and if found increment the environment variable by 1.
`for /f %%j in ('findstr /i /c:"NOTE: DATA STEP STOPPED" "%LogFldr%\%pgm%.log"') do @set /A TtlNumMsc+=1`	Loop through each line in the log looking for the **exact** phrase (**/c:**) "NOTE: DATA STEP STOPPED" which is case-insensitive (**/i**) and if found increment the environment variable by 1.
`for /f %%j in ('findstr /i /c:"STATEMENT NOT EXECUTED" "%LogFldr%\%pgm%.log"') do @set /A TtlNumMsc+=1`	Loop through each line in the log looking for the **exact** phrase (**/c:**) "STATEMENT NOT EXECUTED" which is case-insensitive (**/i**) and if found increment the environment variable by 1.
`for /f %%j in ('findstr /c:"W.D format too small" "%LogFldr%\%pgm%.log"') do @set /A TtlNumMsc+=1`	Loop through each line in the log looking for the **exact** phrase (**/c:**) "W.D format too small" (case sensitive) and if found increment the environment variable by 1.
`for /f %%j in ('findstr /i /c:"Division by zero detected" "%LogFldr%\%pgm%.log"') do @set /A TtlNumMsc+=1`	Loop through each line in the log looking for the **exact** phrase (**/c:**) "Division by zero detected" which is case-insensitive (**/i**) and if found increment the environment variable by 1.

Table 2: Samples of SET and FOR in the Check Log Script

Notice that when searching for the log messages to write to the temporary file, "__logck_%pgm%.txt", it only searched for ERROR: (i.e., included the colon) but when counting the number of messages we wanted to count all the _ERROR_ so that those could be cleaned up as well but we did not want each one of the _ERROR_ written to the external file. This functionality can be changed by updating the FINDSTR option when writing to the external file by removing the colon.

For a complete list of possible log messages being counted refer to the program found on https://github.com/rwatson724/CMD-CheckLog.

Execution Time

Although the execution time of the program is not necessary it can be useful to ensure that you are examining the most current log. Using the STTIME environment variable that was initially set (refer to Environment Variables section) and the current time when the program has completed execution and the log check has completed (ENTIME).

```
set entime=%time%
```

Note that for the environment in which this was developed the time is captured as HH:MM:SS.ss and this

needs to be converted to a number in order to determine the elapsed time. Both STTIME and ENTIME are converted to a number (START and END respectively) using the following logic.

```
for /F "tokens=1-4 delims=:.," %%a in ("%entime%") do (
  set /A "end=(((%%a*60)+1%%b %% 100)*60+1%%c %% 100)*100+1%%d %% 100"
)
```

After the numeric representation of the start and end times are determined, then the elapsed time environment variable can be created and can be converted back to the HH:MM:SS.ss format.

```
set /A "elapsed=end-start"
set /A "hh=elapsed/(60*60*100), rest=elapsed%%(60*60*100),
mm=rest/(60*100), rest%%=60*100, ss=rest/100, cc=rest%%100"
if %mm% lss 10 set mm=0%mm%
if %ss% lss 10 set ss=0%ss%
if %cc% lss 10 set cc=0%cc%
```

Summary Report

The start time, end time and elapsed time can be written to the temporary file, "__logck_summary_%pgm%.txt", followed by the summary of the log messages. Either a message of "No XXXX message found in the log file" or "Total number of XXXX messages in the log file: ##" is displayed in the summary report for each log message type (see the following code snippet).

```
if "!TtlNumERR!" gtr "0" (
  ECHO. Total number of ERROR messages in the log file: !TtlNumERR!
    >> "%LogCk%\__logck_summary_%pgm%.txt"
  ECHO.
    >> "%LogCk%\__logck_summary_%pgm%.txt"
) else (
  ECHO. No ERROR message found in the log file.
    >> "%LogCk%\__logck_summary_%pgm%.txt"
  ECHO.
    >> "%LogCk%\__logck_summary_%pgm%.txt"
)
```

Update Log File

All the unwanted log messages are saved in the temporary file "__logck_ %pgm%.txt" and the summary report is saved in the temporary file "__logck_summary_%pgm%.txt". These reports can then be prepended to the existing log file so that a summary of the log scan is at the top of the log file. To prepend the existing log file, the log file needs to be copied to a temporary file, "__TEMP_%pgm%.log". After it is copied, then the summary report, the list of unwanted log messages and the temporary log file can all be concatenated into one file and given the name of the original log file. After the concatenation of the three files, the temporary files should be deleted.

```
COPY "%LogFldr%\%pgm%.log"           "%LogFldr%\__TEMP_%pgm%.log"
COPY "%LogCk%\__logck_summary_%pgm%.txt" + "%LogCk%\__logck_%pgm%.txt"
+ "%LogFldr%\__TEMP_%pgm%.log"  "%LogFldr%\%pgm%.log"

DEL "%LogFldr%\__TEMP_%pgm%.log"
DEL "%LogCk%\__logck_summary_%pgm%.txt"
DEL "%LogCk%\__logck_%pgm%.txt"
```

If the desire is to have the summary report and the list of log messages as standalone files, then this step can be skipped.

SAS Log 1 and SAS Log 2 show snippets of the updated log file with the summary report showing the counts of unwanted log messages by type as well as a list of the unwanted log messages below the summary. Note that a log message can appear more than once in the “Log scanning – messages are displayed by decreasing order of criticality:” if the message flagged on more than one key word or phrase.

```
Batch execution start at 18:25:10.16
LOG=C:\Users\gonza\OneDrive - datarichconsulting.com\Desktop\GitHub\CMD-CheckLog\
LST=C:\Users\gonza\OneDrive - datarichconsulting.com\Desktop\GitHub\CMD-CheckLog\
Program:     CMD-CheckLog-Test
'CMD-CheckLog-Test' Execution completed at Wed 03/25/2026 18:25:14.73
 ____________________________________

           Time of Execution
 ____________________________________

                 HH:MN:SC,CS
  Start         : 18:25:10.14
  End           : 18:25:18.56
 ------------------------------------
  Elapsed time  : 0:00:08,42
 ____________________________________


 No ERROR message found in the log file.

 No WARNING message found in the log file.

 No uninitialized message found in the log file.

 No MERGE message found in the log file.

 No MISSING VALUES message found in the log file.

 No CONVERT message found in the log file.

 No INFO message found in the log file.

 No miscellaneous unwanted message found in the log file.


--- Log scanning - messages are displayed by decreasing order of criticality:


============================================================ batch END ============================================================
1                                       The SAS System            18:25 Wednesday, March 25, 2026

NOTE: Unable to open SASUSER.PROFILE. WORK.PROFILE will be opened instead.
NOTE: All profile changes will be lost at the end of the session.
NOTE: Copyright (c) 2025 by SAS Institute Inc., Cary, NC, USA.
NOTE: SAS (r) Proprietary Software 9.4 (TS1M9)
      Licensed to MIDWEST SAS USERS GROUP INC, Site 70342576.
NOTE: This session is executing on the X64_WIN+PRO  platform.
```

SAS Log 1: Snippet of Log File with Summary Report – No Unwanted Log Messages

```
Batch execution start at 18:55:08.17
LOG=C:\Users\gonza\OneDrive - datarichconsulting.com\Desktop\GitHub\SAS-Code-WorkSpace\FCMP-ISO Dates\ISO Dates - Updates\\Logs
LST=C:\Users\gonza\OneDrive - datarichconsulting.com\Desktop\GitHub\SAS-Code-WorkSpace\FCMP-ISO Dates\ISO Dates - Updates\
Program:     FCMP_TEST
'FCMP_TEST' Execution completed at Wed 03/25/2026 18:55:12.50
 ________________________________________

           Time of Execution
 ________________________________________

                  HH:MN:SC,CS
   Start         : 18:55:08.15
   End           : 18:55:16.86
 ----------------------------------------
   Elapsed time  : 0:00:08,71
 ________________________________________


 Total number of ERROR messages in the log file: 23

 Total number of WARNING messages in the log file: 10

 No uninitialized message found in the log file.

 No MERGE message found in the log file.

 No MISSING VALUES message found in the log file.

 No CONVERT message found in the log file.

 No INFO message found in the log file.

 Total number of miscellaneous unwanted messages in the log file: 16


--- Log scanning - messages are displayed by decreasing order of criticality:
44:ERROR: Library FCMP does not exist.
45:ERROR: Library FCMP does not exist.
58:ERROR: Library FCMP does not exist.
59:ERROR: Cannot open data set fcmp.funcs for Write access because it is currently opened or already
72:ERROR: Library FCMP does not exist.
73:ERROR: Library FCMP does not exist.
78:ERROR 68-185: The function ISODTTM is unknown, or cannot be accessed.
83:NOTE: Due to ERROR(s) above, SAS set option OBS=0, enabling syntax check mode.
150:ERROR: Library FCMP does not exist.
151:ERROR: Library FCMP does not exist.
156:ERROR 251-185: The subroutine DTTMFMT is unknown, or cannot be accessed. Check your spelling.
164:ERROR 68-185: The function ISODTMPT is unknown, or cannot be accessed.
197:ERROR: Library FCMP does not exist.
198:ERROR: Library FCMP does not exist.
203:ERROR 251-185: The subroutine ZFILL is unknown, or cannot be accessed. Check your spelling.
219:ERROR 251-185: The subroutine ZFILL is unknown, or cannot be accessed. Check your spelling.
229:ERROR 68-185: The function ISODTMPT is unknown, or cannot be accessed.
259:ERROR: Library FCMP does not exist.
260:ERROR: Library FCMP does not exist.
265:ERROR 251-185: The subroutine ZFILL is unknown, or cannot be accessed. Check your spelling.
283:ERROR 251-185: The subroutine ZFILL is unknown, or cannot be accessed. Check your spelling.
291:ERROR 68-185: The function ISODTMPT is unknown, or cannot be accessed.
303:ERROR: Errors printed on pages 1,2,3,4,5,6.
46:WARNING: Unable to load prototypes from 'fcmp.funcs'.
61:WARNING: The following functions will NOT be saved:
74:WARNING: Unable to load prototypes from 'fcmp.funcs'.
85:WARNING: The data set WORK.DATES2 may be incomplete.  When this step was stopped there were 0
152:WARNING: Unable to load prototypes from 'fcmp.funcs'.
171:WARNING: The data set WORK.IND_COMP_DT2 may be incomplete.  When this step was stopped there were
199:WARNING: Unable to load prototypes from 'fcmp.funcs'.
234:WARNING: The data set WORK.DT_ZERO2 may be incomplete.  When this step was stopped there were 0
261:WARNING: Unable to load prototypes from 'fcmp.funcs'.
296:WARNING: The data set WORK.DT_ZERO4 may be incomplete.  When this step was stopped there were 0
82:NOTE: The SAS System stopped processing this step because of errors.
170:NOTE: The SAS System stopped processing this step because of errors.
233:NOTE: The SAS System stopped processing this step because of errors.
295:NOTE: The SAS System stopped processing this step because of errors.
```

SAS Log 2: Snippet of Log File with Summary Report – Unwanted Log Messages

RIGHT-CLICK MENU ITEM

To achieve the optimal functionality of this script, it is best to add it as a right-click menu item. In order to add a right-click menu item, the registry needs to be updated (Schmidt, n.d.; Watson & Hadden, Going Command(o): Power(Shell)ing Through Your Workload, 2023). The following would be saved in a '.reg' file with the path pointing to the location where the __run_checklog.bat file is saved. Notice the use of \\ instead of \ in the path name.

```
Windows Registry Editor Version 5.00


[HKEY_CLASSES_ROOT\*\shell\SAS Batch-Check Log\command]
@="\"C:\\Users\\gonza\\OneDrive -
datarichconsulting.com\\Desktop\\GitHub\\CMD-
CheckLog\\__run_checklog.bat\" \"%L\""
```

Updating the registry is something that an administrator would need to do. If the administrator does not want to update the registry, then the script can still be used via the drag-and-drop. This requires the '.bat' file to be saved in the same location as the program so that the program can be dragged on top of the '.bat' file in order to open the program with the '.bat' file (Image 1).

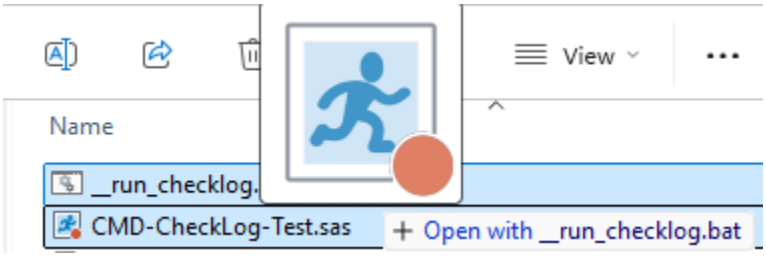

Image 1: Opening SAS Program with a Drag-and-Drop

CONCLUSION

Secondary log checking programs are ideal when running several programs and you want to check all the logs at once. However, during development, running this secondary program can be bit tedious and is sometimes overlooked. The command line interface (CLI/CMD) can greatly improve the review process of program logs during development. There is no longer a need to manually scan through each log or a need to call a secondary log checking program to ensure there are no unwanted log messages.

Full code for the script is available on GitHub: https://github.com/rwatson724/CMD-CheckLog.

Disclaimer: Not all companies allow their employees to access CLI/CMD or to update the registry. It is recommended you work with your IT department to permit curated access to these tools or develop the script or right-click menu item.

REFERENCES AND RECOMMENDED READINGS

Schmidt, R. (n.d.). *Windows: How to add batch-script action to Right Click menu*. Retrieved Nov 2022, from StackExchange: https://superuser.com/questions/444726/windows-how-to-add-batch-script-action-to-right-click-menu

Sheppard, S. (n.d.). *An A-Z Index of Windows CMD commands.* Retrieved Aug 2025, from https://ss64.com/nt/

Sheppard, S. (n.d.). *FINDSTR*. Retrieved Aug 2025, from https://ss64.com/nt/findstr.html

Sheppard, S. (n.d.). *FOR*. Retrieved Jan 2026, from https://ss64.com/nt/for.html

Sheppard, S. (n.d.). *How-to: Pass Command Line arguments (Parameters) to a Windows batch file*. Retrieved Aug 2025, from https://ss64.com/nt/syntax-args.html

Sheppard, S. (n.d.). *How-to: Windows Environment Variables*. Retrieved Aug 2025, from https://ss64.com/nt/syntax-variables.html

Sheppard, S. (n.d.). *SET*. Retrieved Jan 2026, from https://ss64.com/nt/set.html

Watson, R. J. (2017, Sep). *Check Please: An Automated Approach to Log Checking.* Retrieved Aug 2025, from DataRich Consulting: https://datarichconsulting.com/wp-content/uploads/2023/01/Checklogs.pdf

Watson, R. J., & Hadden, L. S. (2023, May). *Going Command(o): Power(Shell)ing Through Your Workload.* Retrieved Aug 2025, from DataRich Consulting: https://datarichconsulting.com/wp-content/uploads/2024/09/Going-Commando_FINAL.pdf

ACKNOWLEDGMENTS

The author wants to thank Lex Jansen for continuing to publish SAS Proceedings from 1976 to present. The website is a source of information for programmers. https://www.lexjansen.com.

The author would also like to acknowledge Mike Goulding and Varun Debbeti for the inspiration of using CMD to perform a log check and to produce a summary report.

The author would also like to thank Rita Lai and Jane Eslinger for their thorough review of paper.

CONTACT INFORMATION

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

Richann Jean Watson

DataRich Consulting

richann.watson@datarichconsulting.com