SESUG Paper 255-2018

Tips and Tricks for Ensuring Accurate Routine Data Processing when Logs are Redirected to Text Files

Davia N. Moyse, ICF

ABSTRACT

SAS® users often develop and maintain a set of codes that process the same expected inputs through to expected outputs and that are run on a routine basis as datasets are updated with new observations. A common efficiency implemented in this data management scenario is to redirect SAS log contents to a permanent text file, rather than display in the terminal log window. This efficiency does provide processing time savings, but may lead to oversight of processing errors when inputs or outputs stray from their expected formats. This paper will discuss techniques that may be implemented to aid a SAS user in reviewing redirected processing logs and the success of processing, especially when no errors or warnings present themselves. Topics will include automatically opening and programmatically reviewing the redirected log text file and checking the intended structure of input, intermittent, and output datasets. Focus will be given to directing some messages back to the terminal log window in the middle of processing, as well as ensuring that all observations are processed through conditional IF-ELSE IF processing. These techniques utilize PROC CONTENTS, PROC SQL, CALL SYMPUT, %SYSEXEC, and %SYSFUNC.

INTRODUCTION

Many SAS® users are responsible for creating and maintaining programs that process data on a routine basis, such as weekly or monthly, for a given project as new observations become available. In this scenario, the typical assumption is that the structure and format of input data and desired outputs are defined upfront and are retained throughout the duration of the project. SAS users can then develop, test, and deploy codes that adhere to the "rule," rather than the "exception."

When SAS data is processed routinely, with programs that are developed to the rule, the SAS user often takes on a role more focused on ensuring and documenting successful processing in a timely manner, rather than programming new codes. One time savings technique is to redirect SAS log contents to a permanent text file, rather than display it in the terminal log window. But, what happens when inputs, intermittent datasets, or outputs contain an exception and processing goes awry? Will the SAS user be able to detect when routine processing has gone off-course if the log has been redirected?

This paper presents solutions that may be implemented to aid a SAS user in reviewing redirected processing logs and the success of processing, including when no errors or warnings present themselves. The first section briefly discusses redirection of SAS logs and the subsequent sections outline three types of solutions:

1. Global Log Review
2. Targeted Review of Datasets
3. Verification of Conditional Processing

The techniques discussed have been developed and implemented in the SAS windowing environment in a Windows operating system, and assumes the user has a basic understanding of SAS' macro facility and various ways to define macro variables (e.g., global assignment, PROC SQL procedure's INTO clause, and DATA step's CALL SYMPUT routine).

REDIRECTING SAS LOGS

To add time savings to running and documenting routine processing, the SAS log can be simply redirected to a permanent file at the top of a SAS program:

```
proc printto log="C:\mypath\week12\routineA.log" new; run;
```

The default behavior of the PROC PRINTTO procedure is to append log data to the external permanent file, if it already exists. The PROC PRINTTO option NEW is used to replace an already existing external log file. The resulting file, whether appended to or a replacement, is viewable in a basic text editor, such as NotePad or TextPad.

Once PROC PRINTTO is invoked, the log contents will only be saved to the permanent file and not shown in the terminal log window. Log contents can be redirected back to the terminal window at any time with a second PROC PRINTTO:

```
proc printto log=log; run;
```

You may alternate between the permanent file and the terminal log window, by specifying different routes in the invoked PROC PRINTTO procedures, in order to document key processing, but still allow targeted checks and messages to be returned to the SAS session for ease of review. Examples of this approach can be found throughout the subsequent sections, where specific checks and messages are discussed.

GLOBAL LOG REVIEW

While redirecting the log to a permanent file has the benefit of automatically documenting the processing (i.e., the user does not need to manually save the log), its downside is that the log contents are not immediately available to the SAS user. One solution is to use the %SYSEXEC macro statement at the end of the program and once the log is directed back to the terminal window. To open the permanent log file in NotePad for SAS user review, use the code:

```
options noxwait;

%sysexec start notepad "C:\mypath\week12\routineA.log";
```

The NOXWAIT system option specifies that you will automatically return to the SAS session once the text editor is opened, and will not be directed to a Windows command prompt window.

In most cases, the basic text editor that you use to automatically open the log file will not retain the colorization that is typical in the SAS windowing environment (i.e., blue is notes, green is warnings, and red is errors). This can make reviewing the auto-saved log more difficult and error-prone. Therefore, a more rigorous solution is to automatically open the log file only after importing it into SAS as a raw flat file to check and flag for the presence of any key error or warning text.

The full global log review concept brings together alternating redirection of the SAS log (A), importing the permanent log file (B), creating a flag for the presence of desired warning or error text (C), populating a message to the log terminal window (D), and automatically opening the log file (E):

```
/* A. Alternating redirection of SAS log */
proc printto log="C:\mypath\week12\routineB.log" new; run;
   <user-defined PROC and DATA steps>
proc printto log=log; run;

/* B. Importing permanent log file */
/* C. Creating flag for any presence of desired warning or error text */
%let errwarn=0;
data _null_;
   infile "C:\mypath\week12\routineB.log" truncover;
   input log_line $200.;
   if index(log_line, 'ERROR:') > 0 or
      index(log_line, 'WARNING') > 0 or
      index(log_line, ' uninitialized.') > 0 then do;
         call symput(compress("errwarn"),compress(1));
   end;
run;

/* D. Populating message to log terminal window */
```

```
%macro msg_global;
   %if &errwarn.=1 %then %put ERROR- an ERROR, WARNING, or UNINITIALIZED
         VARIABLE was encountered during processing;
   %else %put Processing completed successfully;
%mend;
%msg_global;

/* E. Automatically open the log */
options noxwait;
%sysexec start notepad " C:\mypath\week12\routineB.log";
```

With this approach, the SAS user can have the processing automatically documented, specify key text strings from the log that indicate additional investigation is needed, review SAS terminal log for information about the presence of these key warning/error text strings, and have the full log already open for a targeted review if a warning or error is encountered.

TARGETED REVIEW OF DATASETS

A SAS user will likely encounter a situation in their programming career where the program executes without errors or warnings, but an exception from a new observation is encountered along the processing stream that impacts the final output product. As routine processing codes are developed to the rule and may be handling large datasets that are too cumbersome to visually inspect, additional automatic checks beyond a global log review further ensure accurate routine processing.

Throughout the subsections below, example code leverages SAS' ability to color specific text that is placed in the log by the user via the PUT statement. Table 1 below provides details on how text is colored and presented in the log. The use of a colon (:) versus a dash (-) following the WARNING or ERROR dictates whether WARNING/ERROR will be prefixed to the user-defined text.

SAS Code	Resulting Log
put WARNING: this is a warning;	WARNING: this is a warning
put WARNING- this is a warning;	this is a warning
put ERROR: this is an error;	ERROR: this is an error
put ERROR- this is an error;	this is an error

Table 1. SAS Log Rendering via PUT Statement

DATASET STRUCTURE CHECK

The dataset structure check validates the variable names, types, and formats against an expected shell dataset. The shell dataset represents the intended, exact dataset structure, but with no observations. The approach includes outputting dataset contents via the PROC CONTENTS procedure for the dataset being tested and the shell (A), merging the two content datasets and outputting any mismatches (B), checking for mismatches (C), and populating a message to the log terminal window (D), while also implementing the alternating log redirection:

```
proc printto log="C:\mypath\week12\routineC.log" new; run;
/* A. PROC CONTENTS of shell and dataset */
proc contents data=work.dataset
              out=work.dataset_cont (keep=name type format) noprint;
run;
proc contents data=work.shell
              out=work.shell_cont (keep=name type format) noprint;
run;

/* B. Merge content datasets and output mismatched */
proc sort data=work.dataset_cont;
```

```
   by name;
run;
proc sort data=work.shell_cont;
   by name;
run;
data work.struc_check;
   merge work.shell_cont (in=s rename=(type=type_s format=format_s))
         work.dataset_cont (in=d);
   by name;
   if s & ^d then do;
         comment = “Variables in shell data, but not in main dataset”;
         output;
   end;
   else if ^s & d then do;
         comment = “Variable in main dataset, but not in shell data”;
         output;
   end;
   else if s & d then do;
         if type_s ^= type or format_s ^= format then do;
               comment= “Data Type or Formats are not Matched”;
               output;
         end;
   end;
run;

/* C. Check for mismatches */
proc sql noprint;
   select count(*) into :strucobs from work.struc_check;
quit;

/* D. Populating a message to the log terminal window */
%macro msg_struc;
   proc printto log=log; run;
   *NOTE: need to use compress(WARN ING) and compress(ERR OR:) to prevent
         false message to SAS terminal log during global log review that
         there is a warning/error;
   %if &strucobs.=0 %then %put %sysfunc(compress(WARN ING))- DATA
         STRUCTURE CHECK PASSES!;
   %else %if &strucobs.>0 %then %put %sysfunc(compress(ERR OR:)) DATA
         STRUCTURE CHECK FAILS!;
   proc printto log="C:\mypath\week12\routineC.log"; run;
%mend;
%msg_struc;
```

The most likely application of this check is against the initial input dataset or final output dataset, but the approach may be used on any intermittent, key dataset along the processing stream.

VARIABLE (COLUMN) COUNT CHECK

The variable, or column, count check validates that a dataset has the expected number of variables. To apply this approach, the SAS user must know the intended variable count. The technique includes opening the dataset via the OPEN system function (A), retrieving the number of variables via the ATTRN system function (B), comparing the variable count against expected to populate a message to the log terminal window (C), and closing the dataset (D), again, while using alternating log redirection:

```
proc printto log="C:\mypath\week12\routineD.log" new; run;
/* A. Open SAS dataset */;
%let dsid=%sysfunc(open(work.dataset));
```

```
/* B. Retrieve number of variables */
/* C. Compare variable count & populate message to log terminal window */
%macro msg_vars;
   proc printto log=log;
   *NOTE: need to use compress(WARN ING) and compress(ERR OR:) to prevent
         false message to SAS terminal log during global log review that
         there is a warning/error;
   %if %sysfunc(attrn(&dsid.,nvars)) ^= 250 %then
         %put %sysfunc(compress(ERR OR:)) Dataset does not have correct
               number of variables!;
   %else %if %sysfunc(attrn(&dsid.,nvars)) = 250 %then
         %put %sysfunc(compress(WARN ING))- Dataset has the correct number
               of variables!;
   proc printto log="C:\mypath\week12\routineD.log"; run;
%mend;
%msg_vars;

/* D. Close the dataset */;
%let rc=%sysfunc(close(&dsid.));
```

This check is most applicable to intermittent and final output datasets, particularly when the intended variable list has been dictated by DROP statements (rather than KEEP statements) or processing involves bringing together multiple datasets of various data structures.

OBSERVATION (ROW) COUNT CHECK

The observation, or row, count check prints the dataset's number of observations for SAS user validation external to automatic processing. Unlike the variable count check, where the number of columns is known upfront, routine processing of new observations means that the expected row count is in constant fluctuation. The technique includes retrieving the number of observations on the dataset into a macro variable either by the CALL SYMPUT statement or the PROC SQL INTO clause (A1 or A2) and printing the result to the SAS terminal log (B):

```
proc printto log="C:\mypath\week12\routineE.log" new; run;
/* A1. Number of observations into macro variable via CALL SYMPUT */
data _null_;
   set work.dataset;
   call symput(compress("nobs"),compress(_N_);
run;

/* A2. Number of observations into macro variable via PROC SQL INTO */
proc sql noprint;
   select count(*) into :nobs from work.dataset;
quit;

/* B. Print number of observations to the terminal log window */
%macro msg_obs;
   proc printto log=log; run;
   *NOTE: need to use compress(WARN ING) to prevent false message to SAS
         terminal log during global log review that there is a warning;
   %put %sysfunc(compress(WARN ING))- &nobs. observations on dataset;
   proc printto log="C:\mypath\week12\routineE.log"; run;

%mend;
%msg_obs;
```

The most practical application for this check is against both the input and output dataset, to ensure that new observations were initially present when compared to the previous data processing run, as well as retained at the end on the final dataset product. The check also supplies important information when passing final datasets on to other data users, including uploading the data to a SQL database.

VERIFICATION OF CONDITIONAL PROCESSING

Most data processing streams, including routine ones designed for large datasets, contain conditional logic and processing. This conditional processing coupled with explicit OUTPUT statements in DATA steps can be negatively impacted by new observations that have “exception” data. Therefore, in addition to checking specific dataset structure elements, it is also critical to verify conditional processing that results in explicit outputs.

Take the example dataset in Table 2 below (work.sexbirth), where IDs are either male or female as their sex at birth, and you want to parse the data into two separate sex datasets.

IDs (variable: ID)	Sex at Birth (variable: sex)
A04	Female
B56	Female
H42	Male

Table 2. work.sexatbirth (n=3)

Suppose the project this dataset relates to specified that sex at birth will always be non-missing as a rule. The simple code you would likely use is:

```
data female male;
   set work.sexbirth;
   if sex=”Female” then output female;
   else if sex=”Male” then output male;
run;
```

Now consider when three new observations are added to the routine processing, two of which are exceptions and have missing data in the sex at birth variable. See Table 3 below (work.sexatbirth2).

IDs (variable: ID)	Sex at Birth (variable: sex)
A04	Female
B56	Female
H42	Male
F66	
T72	Male
K81	

Table 3. work.sexatbirth2 (n=6)

If you ran the same code as above, you would lose the F66 and K81 observations. A solution to alert the SAS user conducting routine processing includes counting the observations processed through the conditional statements using a sum statement, comparing the total conditionally processed count to the count on the input file, and listing in the log the missed IDs:

```
proc printto log="C:\mypath\week12\routineE.log" new; run;
data female male;
   length id2 $100;
   set work.sexatbirth2 end=eof;
   retain id2;
   if sex=”Female” then do;
         counter + 1;
         output female;
```

```
      end;
      else if sex=”Male” then do;
            counter + 1;
            output male;
      end;
      else do;
            id2 = catx(“ “,id2,id);
      end;
      call symput(‘counter’,compress(counter));
      call symput(‘skippedIDs’,id2);
      if eof then call symput(‘inputN’,compress(_N_));
   run;
   %macro msg_condit;
      proc printto log=log; run;
      *NOTE: need to use compress(ERR OR:) to prevent false message to SAS
            terminal log during global log review that there is an error;
      %if &inputN.^=&counter. %then %do;
            %put %sysfunc(compress(ERR OR:)) IDs skipped during conditional
                  processing + explicit output;
            %put %sysfunc(compress(ERR OR))- Skipped IDs are &skippedIDs.;
      %end;
      proc printto log="C:\mypath\week12\routineE.log"; run;
   %mend;
%msg_condit;
```

The user would then see the following in their SAS terminal log:

```
ERROR: IDs skipped during conditional processing + explicit output
       Skipped IDs are F66 K81
```

CONCLUSION

Various tools and techniques exist to make routine data processing time efficient and ensure that resulting data products are accurate. This paper focused on alternating between a permanent log file and the SAS terminal log to target and report on key data quality checks. The data quality checks presented allow a SAS user to pinpoint where data may stray from the expected format and would otherwise not produce an error or warning.

ACKNOWLEDGMENTS

I would like to thank and acknowledge Wen Song for her encouragement and advice in creating these checks for routine data processing.

CONTACT INFORMATION

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

Davia N. Moyse
ICF
301-572-0295
Davia.Moyse@icf.com