

How to use UNIX commands in SAS code to read SAS logs

James Willis, OptumInsight

ABSTRACT

Reading multiple logs at the end of a processing stream is tedious when the process runs on a UNIX platform. In UNIX, the logs must be individually viewed using a UNIX command like “CAT” or “MORE” or an application like ULTRA EDIT or the UNIX VI edit process. The UNIX options are tedious and time consuming for even skilled UNIX users. The risk of missing an important log message is significant.

SAS batch programs running in UNIX can be chained together using include statements. When a long chain of programs is executed, and each program in the chain has its log printed separately, reviewing all of the logs from the process can be difficult. The process is definitely time consuming. Reading the logs using UNIX commands and SAS logic simplifies and speeds up the log verification process. "ERROR:", "WARNING:", "CONVERSION", "MISSING", "NOT FOUND" and all other messages can be located, ranked, written to a spreadsheet, and then the spreadsheet can be emailed to you.

INTRODUCTION

OptumInsight will execute SAS processes on a UNIX platform chaining multiple programs together so that they use one parameter file. Each program creates its own log file. One of my processes executes over 25 programs creating over 25 logs. I built a SAS program that reads each log created by the process so that I do not have to read each log manually.

CREATE THE MACROS AND LIST OF LOGS TO BE READ

The code uses macros and macro variables to allow for flexibility when programs are not run or extra programs are run during off cycles. Each macro is built by asking the question “Is it code or is it data?”. First you list the essential names of the log files that will be read:

```
%LET loglist = 's050_prerun_format_testing',
               's100_crm',
               's100_online_reg',
               's100_gym_activation',
               's100_event_import',
               's100_c360',
               's100_ICUE_Coaching',
               's150_crm_to_lu_ayb',
               's200_file_finder',
               's300_LU_AYB_matching',
               's400_QA_Import_Files',
               's500_lu_ayb_indv_id',
               'rs003_null_values_by_table',
               'rs02_unspecified_counts',
               'rs01_NOVU_Report',
               's600_copy_files_to_archive'
               ;
```

The order of the names is not important except for the first and last names. My code lists the names in the order that they are included by the execution process. I use %GLOBAL statements for each macro defined in the code.

```
%let projekt = AYB_WEEKLY_LOGS_REVIEW;
options lrecl=256 linesize=256 spool nomprint nosymbolgen nomlogic;
```

Create a macro that contains the date that the logs were created on and have as their suffix. NamDt is used in the name of the log. RunDt is used in the name of the folder where the logs are stored.

```
%let offset = 0; /** change here **/

data _null_ ;
format namdt $7. f 8.;
e = today() - %eval(&offset.) ;
rundt = put(e,yymmddn8.) ;
call symputx('RunDt',rundt) ;
f = "&sysdate."d - %eval(&offset.) ;
namdt = put(f,date7.);
call symputx('NamDt',namdt) ;
run;
```

Create a macro that emails the results of the log review, created as a spreadsheet. The data null changes the UNIX location to the location of the excel file being emailed. My code creates an EXCEL .xlsx file.

```
%macro emailit;
data _null_ ;
%SYSEXEC %STR(cd /unix/folder/name/and/file/location;);
run;

filename mymail email
subject="Place your Email subject words here"
TO="email.recipient@mail.address"
attach("&logdir./&projekt..xlsx" content_type="application/vnd.ms-excel" LRECL=9999)
;
run;

data _null_ ;
file mymail;
put ' ';
put "The logs for &projekt. in &logdir. have been reviewed";
put ' ';
put "Check the attached file thoroughly to find any problems.";
put ' ';
put "The attached file contains items that may be significant.";
put 'The file also contains observation counts of datasets created during the process. ';
put ' ';
run;
%mend emailit;
```

CREATE THE MAIN LOGIC

The logchek macro works for every file named in the &loglist. macro. Each line in the log is read, ranked, and classified. A line with chekline = 0 is not sent to the spreadsheet. The output file is sorted in rank order.

```
%macro logchek;
proc sql;
    drop table work.&logshrt.;
quit;

data work.&logshrt.;
    length row 8 chekline 8 source $ 30 message $ 25 logline $256;
    infile logchek linesize=256 recfm=v trunccover pad noprint END=FINI;
    input logline $char256.;
    logline = strip(left(upcase(logline)));
    row = _n_;
    if row = 1 then do;
        chekline = 5;
        source = trim(left("&logpre."));
        message = '';
    end;
    chekline = 0;
    source = trim(left("&logpre."));
    message = '';
    if index(logline,'ERROR!') > 0 and index(logline,'MPRINT') = 0 then do;
        chekline = 1;
        message = " FOUND AN ERROR!";
        call symput(cat('ERR'||strip(put(row,8))),put(row,8.));
    end;
    if index(logline,'ERROR:') > 0 then do;
        chekline = 1;
        message = " FOUND AN ERROR:";
        call symput(cat('ERR'||strip(put(row,8))),put(row,8.));
    end;
    if index(logline,'WARNING:') > 0 then do;
        message = "FOUND A WARNING:";
        chekline = 2;
        IF index(logline,'HAS NOT BEEN DROPPED') > 0 then chekline = 9;
        IF index(logline,'.DATA DOES NOT EXIST') > 0 then chekline = 7;
    end;
    if index(logline,'NUMERIC') > 0 then do;
        message = "NUMERIC OR CHARACTER CONVERSION OCCURRED";
        chekline = 2;
    end;
    if index(logline,'MISSING') > 0 then do;
        message = "FOUND A MISSING VALUE";
        chekline = 3;
    end;
    /** division by 0 detected **/
    if index(logline,'DIVISION') > 0 then do;
        message = "FOUND A DIVISION BY ZERO";
        chekline = 4;
    end;
    if index(logline,'UNINITIALIZED') > 0 then do;
```

```

        message = "UNINITIALIZED VARIABLE";
        chekline = 5;
    end;
    if index(logline,'show stopper') > 0 and index(logline,'MPRINT') = 0
    then do;
        message = " FOUND A SHOW STOPPER" ;
        chekline = 1;
    end;
    if index(logline,'NOTE') > 0 and index(logline,'STOPPED') > 0
    then do;
        message = " FOUND AN ERROR MESSAGE" ;
        chekline = 1;
    end;
    if index(logline,'=== =====>') > 0 and index(logline,'MPRINT') = 0
    then do;
        message = " FOUND AN ARROW:" ;
        chekline = 2;
        if index(logline,'+PUT') > 0 then do;
            message = " " ;
            chekline = 0;
        end;
    end;
    if index(logline,'NOTE') = 0 then do;
        if index(logline,'+') > 0 then chekline = 0;
        if index(logline,'+') > 0 then chekline = 0;
        if index(logline,'+RUN;') then chekline = 0;
    end;
    /** print notes unless specifically not printed ***/
    if index(logline,'NOTE:') > 0 and chekline = 0 then do;
        if index(logline,'DROPPED') > 0 then chekline = 0;
        if index(logline,'PROCEDURE SQL') > 0 then chekline = 0;
        if index(logline,'The data set') > 0 then chekline = 9;
        if index(logline,'THE DATA SET') > 0 then chekline = 9;
        if index(logline,'OBSERVATIONS') > 0 then chekline = 9;
        if index(logline,'REAL TIME') > 0 then chekline = 0;
        if index(logline,'CPU TIME') > 0 then chekline = 0;
        if index(logline,'RUN;') > 0 then chekline = 0;
        if index(logline,'PRINTTO') > 0 then chekline = 0;
        if index(logline,'BEGINS') > 0 then chekline = 0;
        if index(logline,'INFO') > 0 then chekline = 8;
        if index(logline,'NDS') > 0 then chekline = 0;
        if index(logline,'MPRINT') > 0 then chekline = 0;
        if (index(logline,'TABLE') > 0 and index(logline,'CREATED;')) then chekline = 6;
        if index(logline,'SYMBOLGEN') > 0 then chekline = 0;
        if index(logline,%str('MPRINT()')) > 0 then chekline = 0;
        if index(logline,%str('(TOTAL PROCESS TIME)')) > 0 then chekline = 0;
    end;
    if index(logline,'THE SAS SYSTEM') > 0 then chekline = 0;
    if index(logline,'NOTE: The SAS System stopped') > 0 then chekline = 1;
        if index(upcase(logline),'STOPPED') > 0 then do;
            message = "ERROR";
            chekline = 1;
        end;
    if index(upcase(logline),'ERROR 180-322') > 0 then do;
        message = "ERROR";
        chekline = 1;
    end;

```

```

        end;
    if chekline > 0 then do;
        output;
        return;
    end;
run;

proc sort data=work.&logshrt.;
by chekline;
run;

%if "&logpre" = "&fstfil." %then %do;
    data work.logslst;
    set work.&logshrt.;
    run;
%end;
%else %do;
    proc append base=work.logslst
        data=work.&logshrt.;
    run;
%end;

%put &=export;
%put "&logdir./&projekt..xlsx";

%if &export. = Y %then %do;
    data _null_;
    %SYSEXEC %STR(cd &logdir.; rm &logdir./&projekt..XLSX);
    run;

    proc sort data=work.logslst;
    by chekline source;
    run;

    proc export
    data=work.logslst
    outfile = "&logdir./&projekt..xlsx"
    replace
    DBMS=xlsx;
    sheet=&projekt.;
    run;
%end;
%mend logchek;

```

EXECUTE THE MAIN LOGIC FOR EACH LOG FILE

The logset macro assigns the macro values to be used by the logchek macro then calls the logchek macro.

```

%macro logset(log1=,export=,logshrt=);
%let logpre = &log1.;          /** change here */
%let base = /location1/location2/location3;  /** change here */
%let pgm = /location4/location5;  /** change here */
%let fldr = log_file_location_&RunDt.;
%let logdir = &base.&pgm.&fldr./location6;  /** change here */

```

```

%let lognam = &logpre._&NamDt.;
%let export = &export.;
%let logshrt = &logshrt.;
%put "logchek lognam = &logdir./&lognam..log";

filename logchek clear;
filename logchek "&logdir./&lognam..log";
%logchek;
%mend logset;

%let frstfil = s050_prerun_format_testing; /* first name in the loglist macro */
%let lstfil = s600_copy_files_to_archive; /* last name in the loglist macro */

/**** This will execute for as many log files as are in the "do" statement ****/
/**** All the log files have to be in the same UNIX folder location ****/

```

Each file named in the loglist macro variable is executed. When the file name in the loglist macro variable is the first log file or the last log file, special processing occurs. The email macro is called only after the last log file has been read.

```

data _null_;
length log $ 100;
exp = 'N';
do log =
&loglist.
;
logshrt = substr(log,1,4);
put "log = " log " and lstfil = &lstfil. ";
if trim(left(upcase(log))) = trim(left(upcase("&lstfil."))) then exp = 'Y';
else exp = 'N';
call execute("%logset(log1='||log||',export='||exp||',logshrt='||logshrt||')");
if exp = 'Y' then do;
call execute("%emailit");
end;
end;
run;

```

SPREADSHEET EXAMPLE

This picture shows an example of how warning messages in the log for the s100_ICUE_Coaching program were ranked and output to the final spreadsheet. Ranking, showing the log source, log row and log message, allows me to quickly find issues that happened during processing. You know right away which program or programs in the process need to be reviewed.

	A	B	C	D	E	F	G	H	I	J	K
1	row	ckeline	source	message	logline						
2	1049	1	s100_ICUE_Coaching	ERROR	WARNING: DATA SET WORK.LU_AYB_NOMATCH WAS NOT REPLACED BECAUSE THIS STEP WAS STOPPED.						
3	1091	1	s100_ICUE_Coaching	ERROR	WARNING: DATA SET WORK.LU_AYB_NOMATCH WAS NOT REPLACED BECAUSE THIS STEP WAS STOPPED.						
4	1142	1	s100_ICUE_Coaching	ERROR	WARNING: DATA SET WORK.ICUE_LU_AYB_MATCH3 WAS NOT REPLACED BECAUSE THIS STEP WAS STOPPED.						
5	1144	1	s100_ICUE_Coaching	ERROR	WARNING: DATA SET WORK.LU_AYB_NOMATCH WAS NOT REPLACED BECAUSE THIS STEP WAS STOPPED.						
6	1192	1	s100_ICUE_Coaching	ERROR	WARNING: DATA SET WORK.ICUE_LU_AYB_MATCH3 WAS NOT REPLACED BECAUSE THIS STEP WAS STOPPED.						

Display 1. Example of the spreadsheet that is created and emailed.

CONCLUSION

Reading multiple logs at the end of a processing stream is tedious when the process runs on a UNIX platform. In UNIX, the logs must be individually viewed using a UNIX command like “CAT” or “MORE” or an application like ULTRA EDIT or the UNIX VI edit process. The UNIX options are tedious and time consuming for even skilled UNIX users. The risk of missing an important log message is significant.

Using SAS to read log files, with an infile and an input statement, allows a programmer to read each line of a log, search for key words in each line, rank the key words found on each line, write out only the key findings, and email the key findings to one or more persons, makes a tedious, time consuming and risky chore, simple, quick and valuable.

REFERENCES

Robbins, Arnold. August 1999. ‘UNIX IN A NUTSHELL’, 3RD Edition. Sebastopol, CA : O’Reilly.

CONTACT INFORMATION

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

James Willis
OptumInsight
763-283-3654
james.willis@optum.com