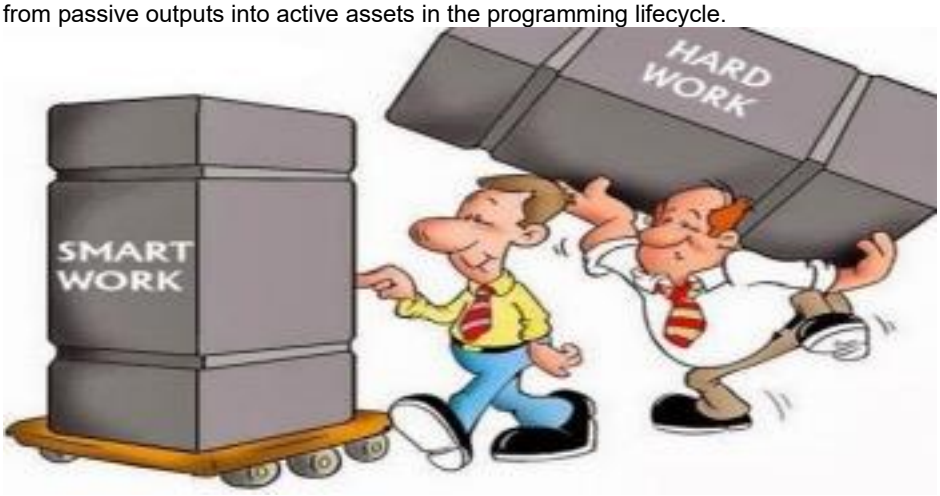
Paper CT12

How SAS logs divert program execution, timeline estimates, data-output dependency and program recovery beyond error and debugging

Avinash Kumar, Eli Lilly, Bengaluru, India

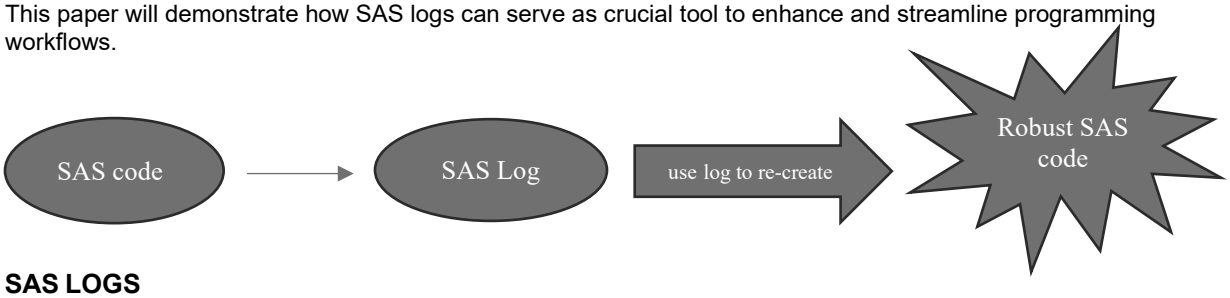
ABSTRACT

SAS language serves as one of most widely utilized statistical analysis programming tool. This paper aims to highlight the multifaceted utility of SAS logs, emphasizing their value beyond error detection. It delves into how SAS logs can be harnessed to divert program execution workflow, estimates timelines near important milestones, identify dependencies between data and their related outputs and even recover lost programs. Through practical examples and real-world applications, this paper aims to highlight how a deeper understanding of SAS logs can transform them from passive outputs into active assets in the programming lifecycle.



INTRODUCTION

SAS (Statistical analysis system) is powerful tool. SAS log is one of the most important entities of it. This helps to debug, to understand program flow, to analyze performance and to know about input data of the program and etc. This paper will demonstrate how SAS logs can serve as crucial tool to enhance and streamline programming workflows.



SAS LOGS

During interactive SAS sessions, a log of executed program is generated. For documentation and reproducibility, it is recommended to save these logs to an external file. This practice is particularly important for the topics discussed in this paper. To redirect the SAS logs to an external file, use the procedure ‘proc printto’ along with the file path. To stop logging to external file and revert to default behavior, simply use a blank proc printto statement. Any SAS code executed between these two statements will have its log saved to specified external file.

```
proc Printto log="/home/userid/saslogs/pgm.log" new;
run;

data class;
set sashelp.cars;
 where origin="Asia";
run;
```

```
proc Printto;
run;
.
```

LOGS DRIVEN EXECUTION

Program execution can be conditionally redirected based on log messages such as notes, warning, or error generated. For instance, I encountered a scenario where we needed to primarily perform multiple imputation using retrieved dropout(RD) approach over return-to-baseline(RTB) approach. This is a main topic in the paper body. The available data was not enough to run on RD approach and leading to log issue and termination of programs. Being unsure of upcoming data which can be anytime fit for RD or RTB approach. We used log driven execution approach.

```
filename logfile "/home/userid/saslogs/pgm/adxxmi.log"; /*main log(example purpose
only)*/
```

Step-1

```
filename logfile "/home/user_x/saslogs/pgm/rd_part.log";

/***--- Start RD module and capture its log--***/
proc Printto log = log_rd new;
run;

%rd_modue; /*imputation using retrieved dropout (example purpose only)*/
proc Printto log=log; /*End of capturing log of RD module*/
run;
```

Step-2

Reading RD module log and searching for message (e.g. 'Not enough observation to fit regression models') that indicates not enough data to apply imputation using RD and store relevant information in macro variable switch. This is specific scenario we faced, This can be generalized for any specific message as per need.

```
data rd_log;
infile "/home/user_x/saslogs/pgm/rd_part.log" dlm='0A'x;
length VAR $500;

do until(VAR=' ');
       input VAR : $500. @;
       output;
end;
run;

data rd_not_met;
set rd_log;
where var ? 'Not enough observations to fit regression models';
run;

proc Sql noprint;
select count(distinct var) into :switch from rd_not_met;
quit;
```

Step-3

If switch macro has non-zero value then it indicates to use the second approach(RTB).

```
%if &switch %then %do;

/*when retrieved dropout did not converge*/
/***--- Start RTB module and capture its log--***/
filename log_rd "/home/user_x/saslogs/pgm/rtb_part.log";

proc Printto log = logfile new;
%module_rtb;/*imputation using return to baseline(example purpose only)*/

proc Printto log=log; /*End of capturing log of RTB module*/
run;
```

Step-4
Combine both the logs to single(final) log file if RD approach fails and RTB works:

```
data rtb_log;
infile "/home/user_x/saslogs/pgm/rtb_part.log" dlm='0A'x;
length VAR $500;

do until(VAR=' ');
       input VAR : $500. @;
       output;
end;
run;

data _null_;
set rd_log rtb_log; /*combined log of rd and rtb */
file "/home/user_x/saslogs/pgm/adxxmi.log";
put var;
file print;
run;

%end;

/*when retrieved dropout converged*/
%else %do;

data _null_;
       set rd_log;
       file "&ir_adlog_lst./&pgmname..log";
       put var;
       file print;
run;

%end;
```

CHECK DEPENDENCY

Logs are valuable tools for identifying dependencies between datasets and outputs. They play a crucial role in assessing the impact of changes made to single dataset by revealing how those changes affect downstream datasets and outputs. Additionally. Logs enhance traceability throughout the data processing workflow.

```
/* Read all files from given location*/
data filename;
length fref $8 fname $200;
did=filename(fref,"/home/userid/saslogs/pgm/");
did=dopen(fref);

do i=1 to dnum(did);
       fname=dread(did,i);
       output;
end;

did=dclose(did);
did=filename(fref);
keep fname;
run;
/* Filter out only logs from the location*/
data all_log;
set filename;
pgm_name=strip(scan(fname,1,'.'));

if index(lowcase(fname),'log');
run;
```

```
/* read all the logs */
proc Sql noprint;
select distinct pgm_name into :pgm_list separated by '*' from all_log;
select count(distinct pgm_name) into :pgm_list_count separated by '*' from all_log;
quit;

%do i=1 %to &pgm_list_count;
%let pgm&i=%sysfunc(scan(&pgm_list,&i,'*'));

data log&i;
       infile "/home/user_x/saslogs/pgm/&&pgm&i...log" dlm='0A'x;
       length VAR $500;

       do until(VAR=' ');
              input VAR : $500. @;
              output;
       end;
run;

/*Find out all input data from SDTM, ADaM, etc. used for each programs*/
data log_d&i;
       set log&i;
       where upcase(var) ? 'SDTM.' Or upcase(var) ? 'ADAM.';
run;
```

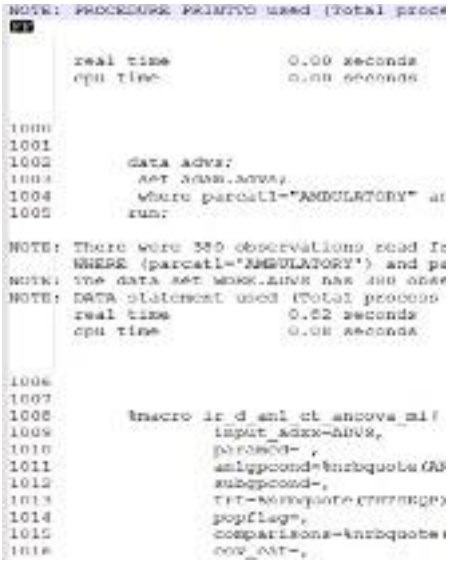
This will help creating table containing output names and their input datasets as below,

output	inp_dslist
bdcv_notable_css	ADHYPO;ADHYSUM;ADAE
lsaemal	AE;ADDV;ADHYPO;ADAE
lscon01	DM;AE;LB;CM;XL;RELRECFL;EX;ADCM
sma1c01	ADA1C;ADA1CMI
smaeh01	ADAE
smaeh02	ADAE
smaek01	ADAE
smaen01	ADAE
smcgm01s_inadv	ADCGM;ADCGMI70
smcgm02_visit	DS;ADCGM

This helps to assess update in 1 data which all outputs need to be checked for possible impact.

RETRIEVE LOST PROGRAMS

SAS logs contain extensive information, including the actual program. This makes it possible to extract those lines from SAS code and quickly recover the lost program from latest executed log file. Refer part of 'check dependency' section for reference program to read log from given location. Below is example of how logs appear with line numbers.



```
data s1(keep=code_part);
set xx_log;
s1=input(scan(var,1,''),??best.);
s1n=input(s1,??best.);
s1n_l=lengthn(strip(put(s1n,best.)));
code_part=left(substr(var,s1n_l+1));

if s1n > .;
run;
```

Then write back recovered program to XX_rec.sas.

```
data _null_;
set s1;
file "&out_path./xx_rec.sas";
put code_part;
file print;
run;
```

TIMELINE ESTIMATION

SAS logs provide insights into the actual time taken for program execution. By reviewing logs across study-related programs, one can estimate the total expected runtime. This information is valuable for planning and scheduling execution, especially when approaching key project milestone. Refer part of ‘check dependency’ section for reference program to read logs from given location.

```
/*Filter out only line with RUN TIME*/
data run_time&i;
set log&i;
where upcase(var) ? ‘RUN TIME’;
run;
```

Note: Run time shows in logs are in HH:MM:SS format. Convert this time to seconds and sum up all second time to get total real time taken for programs.

Similarly total CPU time can be calculated. Repeat this task for all available SAS programs to estimate total time taken required for execution.

CONCLUSION

SAS logs offer more than just error detection; they facilitate workflow automation and program recovery. Analyzing logs boosts team efficiency, ensures compliance, and reduces risks. Treating logs are as dynamic resources enhances every stages of programming and project delivery.

CONTACT INFORMATION

Your comments and questions are valued and encouraged.
Contact the author at:
Author Name: Avinash Kumar
Company: Eli Lilly India
Address: Bengaluru, India
Work Phone: 08046640990
Email: kumar_avinash@lilly.com
Website: www.lilly.com