PharmaSUG 2026 - Paper 313

Having Your Cake and Eating It Too: Automated Log Analysis Without Losing the SAS EG Log Window

Steve Black M.S.P.H, Neurocrine Biosciences Inc.

ABSTRACT

For many SAS programmers, the SAS® Enterprise Guide (EG) log presents a familiar challenge: while the log is fully visible in the EG interface, it is difficult to programmatically scan for subtle but important conditions such as uninitialized variables, problematic MERGE BY statements, and implicit type conversions. Redirecting the log to an external file enables automated review but removes it from the EG log window, forcing an uncomfortable trade-off.

This paper demonstrates that with a little fancy footwork, we can have our cake and eat it too! Leveraging SAS Enterprise Guide's ability to execute custom code before and after program submission, the SAS log can be temporarily redirected to a physical file for automated analysis and then seamlessly restored and replayed into the EG log window. This approach preserves the interactive EG log experience while enabling robust, programmatic log checks. In addition, the solution also generates structured HTML reports, including consolidated summaries when multiple programs are run.

INTRODUCTION

The SAS log is a critical diagnostic tool, often revealing issues that may not stop execution but still indicate underlying problems. In SAS EG, however, interacting with the log in a programmatic way can be more restrictive than expected. While the EG log window provides visibility into program execution, it limits the ability to automatically scan the log for conditions such as uninitialized variables, problematic MERGE BY statements, and numeric to character conversions.

A common workaround is to redirect the log to an external file for post-processing. Although this enables automated review, it comes at the cost of losing log visibility within the EG interface. As a result, programmers are often forced to choose between an interactive log or a searchable one.

This paper presents an approach that removes this trade-off. By using SAS Enterprise Guide's pre- and post-submission execution features, the SAS log can be temporarily redirected to a physical file for automated analysis and then restored and replayed into the EG log window. This allows programmers to retain the familiar EG log experience while enabling robust, automated log review.

OVERVIEW OF THE APPROACH

The solution can be broken into a small number of steps that build on one another.

At a high level, the approach consists of the following:

1. Redirect the SAS log to a physical file before program execution
2. Execute the user's SAS code normally
3. Restore control of the log to SAS Enterprise Guide
4. Replay the captured log back into the EG log window
5. Scan the physical log file for conditions of interest
6. Summarize the results in a reviewable report

The sections that follow walk through each of these steps using simplified code examples. Providing a solid framework where additional logic to the program can be applied including additional error handling and reporting features.

REDIRECTING THE SAS LOG BEFORE EXECUTION

The first step is to capture the SAS log in a physical file. This is accomplished using PROC PRINTTO in code that executes prior to program submission in SAS Enterprise Guide.

```
/* Pre-submission code */
filename eglog "X:\temp\example.log";
proc printto log=eglog new;
run;
```

Pre-submission Code

This basic approach works; however, it can be extended to dynamically name the log file based on the program being executed. Ideally, the log file should:

- Have the same name as the program
- Use a .log extension
- Be stored in the same directory as the program

This can be achieved using the automatic macro variable &_SASPROGRAMFILE, which is populated when a program is saved. If the program has not yet been saved, this value will be blank. In that case, a temporary file can be created in the WORK directory using a default name such as program_neweditor.sas.

Example output of the %put &_sasprogramfile.; code from an unsaved program.

"F:\SAS Temporary Files_TD30184_USW1PRDSASXX_\Prc2\program_neweditor.sas"

The following macro implements this logic and creates three global macro variables (seefile, seefolder, and log_file) used throughout the process.

```
/* Pre-submission code */
%macro precode;
%global seefile seefolder log_file;
%if &_sasprogramfile. = '' %then %let _sasprogramfile =
"%sysfunc(pathname(work))\program_neweditor.sas";
%put &_sasprogramfile.;
%let seefile=%substr(&_sasprogramfile,2,%eval(%length(&_sasprogramfile)-2));
%let log_file=%substr(&seefile,1,%eval(%length(&seefile)-4)).log;

/*** obtain the folder path and adjust per company folder names ***/
data _null_;
    length seefolder log_file $200;
    seefolder = substr("&seefile", 1, findc("&seefile", '\', -length("&seefile")));
    seefolder = tranwrd(seefolder, 'X:', '&_rootpath');
    seefolder = substr(seefolder, 1, length(seefolder)-1);
    call symputx('seefolder', seefolder);

/*** Optional code ***/
    log_file = tranwrd("&log_file", 'X:', '&_rootpath');
    call symputx('log_file', log_file);
run;

/*** change the log to be saved in the specified location ***/
filename log_file "&log_file.";
proc printto log=log_file new;
run;

%mend;

%precode;
```

Pre-submission Macro

In some environments, mapped drives (such as X:) may resolve differently on server infrastructure. In this example, the path is adjusted using the &_rootpath macro variable. This step may not be necessary in all environments.

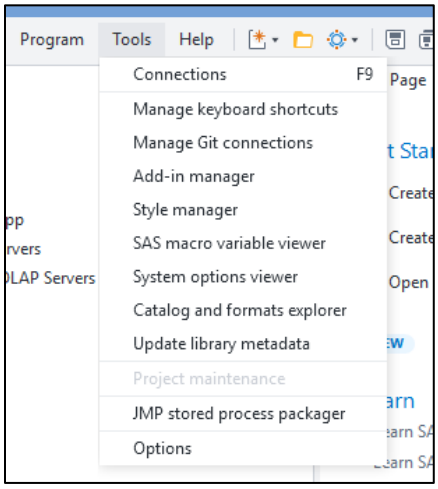
I have set this code up so that I have both the pre-submission code and the stored in a single file saved in a permanent location and then I reference the location using an include statement and run the macro code.

Once defined, this macro (%precode) and post submission code (%saseg_logcheck) can be stored in a single file and referenced using an %INCLUDE statement. The %precode macro call can then be inserted into SAS Enterprise Guide as such:

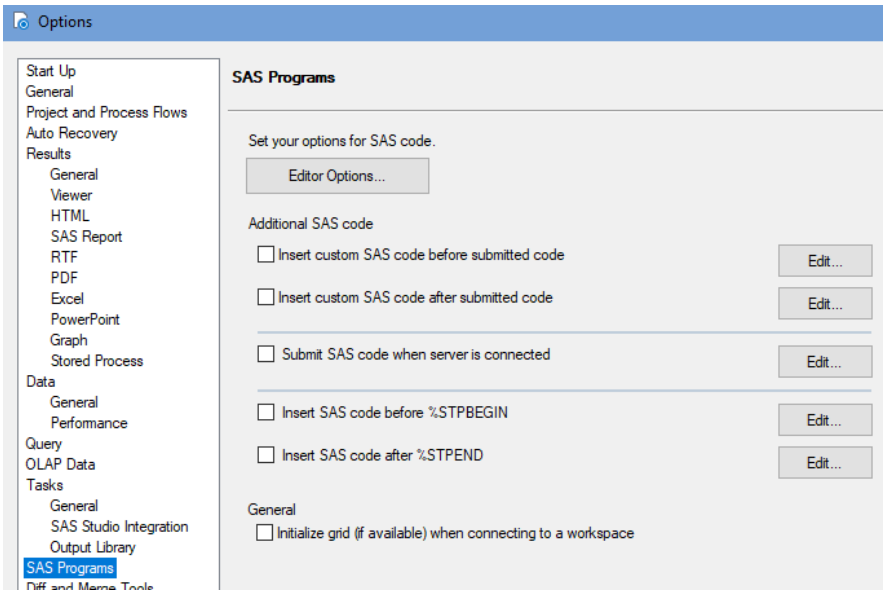
```
/* Pre-submission code */
%include "<your path>\saseg_logcheck.sas";
%precode;
```

Pre-submission Code

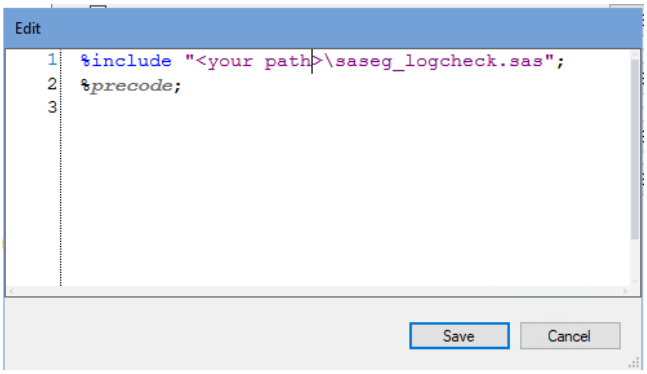
This code will be saved in the Tools > Options > SAS Programs Tab within the Edit box for the “Insert custom SAS code before submitted code”.



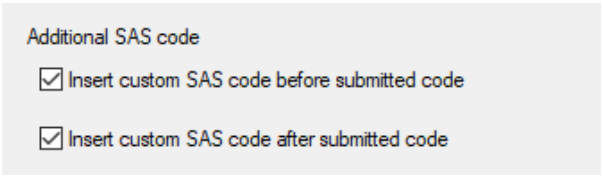
Accessing the Options Menu



The SAS Programs Menu



Inserting Code in the Edit Box



Selecting the Before and After Submitted Code Option

EXECUTE THE USER'S SAS CODE NORMALLY

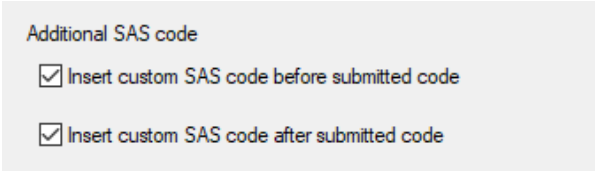
At this point, all log output generated by the program is written to the specified file. Although the log is temporarily not visible in the EG log window during execution, this is intentional and will be corrected in the next step.

RESTORING AND REPLAYING THE LOG IN SAS ENTERPRISE GUIDE

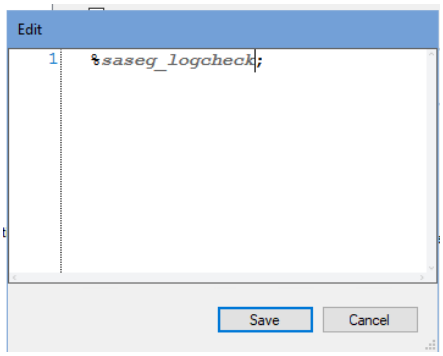
After the user's program completes, control of the log is returned to SAS Enterprise Guide and the captured log is replayed into the EG log window.

This is accomplished using a post-submission macro, which can be configured under:

Tools > Options > SAS Program within the "Insert custom SAS code after submitted code".



Selecting the Before and After Submitted Code Option



Inserting the %saseg_logcheck Macro Call in the After Submitted Code Edit Box.

We can build the post submission code (SASEG_LOGCHECK macro) within the same program that we saved the PRECODE macro. The first step of this macro is to grab the contents of the captured log file and write them back into the EG log window so that the log appears as if it had never been redirected.

```
/* Post-submission code */

%macro saseg_logcheck;

proc printto log=log;
run;

/* Save the log file to the location specified */
data _null_;  infile log_file; input; put _infile_; run;
proc printto;
run;
```

Begin Post Submission Code (%saseg_logcheck macro)

The call to PROC PRINTTO LOG=LOG restores the default log destination. Reading the log file and writing each line with a PUT statement causes the log content to be re-emitted into the EG log window. From the programmer's perspective, the log remains fully visible and behaves as expected.

SCANNING THE LOG FOR CONDITIONS OF INTEREST

Once the log has been captured in a physical file, it can be treated as plain text and scanned programmatically. In this step we create a program macro variable which is the program name of the log that we are reading in. A simple data step can be used to identify common conditions of interest.

```
%let program = %scan(%scan(&seefile,-1,\),1,.);

data log_messages;
    length program type $20 message $2000;
    infile log_file truncover;
    input message $2000.;

    if upcase(message) =: 'ERROR' then type='ERROR';
    else if upcase(message) =: 'WARNING' then type='WARNING';
    else if index(upcase(message),'UNINITIALIZED') then type='UNINITIALIZED';
    else if index(upcase(message),'MERGE') and index(upcase(message),'BY') and
    index(upcase(message),'REPEATS') then type='MERGE BY ISSUE';
       else if index(upcase(message),'CONVERTED TO NUMERIC') then type='CONVERTED TO
NUMERIC';
       else if index(upcase(message),'CONVERTED TO CHARACTER') then type='CONVERTED TO
CHARACTER';

      program="&program";

      if type ne '';

run;

filename log_file clear;
```

Bringing in the Log File and Scanning for Issues

This example illustrates the basic idea: the SAS log is read line by line, and simple string logic is used to classify messages. Additional logic can be added to detect other conditions such as empty data sets.

HANDLING WRAPPED AND FRAGMENTED LOG MESSAGES

Although uncommon, log messages may occasionally wrap across multiple lines, making simple pattern matching insufficient. In these cases, adjacent lines can be evaluated using the LAG function.

```
/* Optional Code */
```

```
    else if (upcase(message)=:'RROR' and strip(lag(message)) = "E") or
(upcase(message)=:'ROR' and strip(lag(message)) = "ER")
      or  (upcase(message)=:'OR' and strip(lag(message)) = "ERR") or
(upcase(message)=:'R' and strip(lag(message)) = "ERRO")
      then type='ERROR';
```

This same approach can be extended to other keywords such as WARNING or ALERT.

SUMMARIZING AND REPORTING RESULTS

After log messages have been classified, the results can be summarized and presented in a structured report.

```
/* Create summary counts by file and message type */
proc sql;
    create table log_summary as
    select program,
           type,
           count(*) as message_count
    from log_messages
    group by program, type;
quit;

/* Generate HTML report */
ods listing close;
ods html path="&seefolder" file="log_summary.html" style=htmlblue;

title1 "SAS Log Summary Report";
title2 "&sysdate9 &systime";

proc report data=log_summary nowd;
    column program type message_count;
    define program / "Log File";
    define type / "Message Type";
    define message_count / "Number of Messages";
run;

title "Detailed Log Messages";

proc report data=log_messages nowd;
    column program type message;
    define program / "Log File";
    define type / "Message Type";
    define message / "Log Message" style(column)=[cellwidth=6in];
run;

ods html close;
ods listing;

/* clean up datasets */
proc datasets lib=work nolist;
        delete log_messages log_summary;
quit;

%mend;
```

Creating a Summary of Log Issues Found

Two summary html reports are created, the first counting the number of issues noted in the program and the second presenting the full message associated with each issue. The logic also cleans up the temporary datasets created minimizing the impact to the program being evaluated.

CONCLUSION

By redirecting the SAS log, replaying it back into the SAS Enterprise Guide log window, and performing automated analysis on the captured file, it is possible to eliminate the traditional trade-off between log visibility and programmatic review. This approach allows SAS programmers to retain the interactive EG log experience while gaining powerful, automated insight into the details that matter most.

ACKNOWLEDGMENTS

The author would like to thank and acknowledge Santhosh Karra for his pivotal work in the development of this approach. His ingenuity and creativity are always greatly appreciated.

CONTACT INFORMATION

Your comments and questions are valued and encouraged.

Steve Black
Neurocrine Biosciences
steven.c.black@gmail.com

APPENDIX

In the appendix I provide full macro code for when a log summary of all programs in a certain folder is needed. This macro loops through all the logs in the specified directory and creates a summary of each log output.

```
%macro saseg_logcheck_all (dir);

   filename progdir "&dir";

   /* Get file list */
   data filelist;
          length fname $200;
          did=dopen("progdir");

          if did>0 then
                 do i=1 to dnum(did);
                        fname=dread(did,i);
                        output;
                 end;

          rc=dclose(did);
          keep fname;
   run;

   %let cur_pgm=%lowcase(%scan(%scan(&seefile,-1,\),1,.).log);
/* %put &cur_pgm;*/

   data log_list;
   set filelist;
          if lowcase(scan(fname,-1,.')) ne 'log' then delete;
          if lowcase(fname)=: 'runall' then delete;
          if index(lowcase(fname),"&cur_pgm") then delete;
   run;

   filename progdir clear;

   proc sql noprint;
          select fname into: fname1 -: fname999
          from log_list;
          %let num_logs=&sqlobs;
   quit;

   %do x = 1 %to &num_logs;
          %if %length(&dir) %then %do;
              filename lg_file "&dir.\&&fname&x";
          %end;

          %let program = %scan(&&fname&x,1,.);

          data _xlog_messages&x;
              length program type $20 message $2000;
              infile lg_file truncover end=eof;
              input message $2000.;

              retain found_issue 0;

              if upcase(message) =: 'ERROR:' then type='ERROR';
              else if upcase(message) =: 'WARNING:' then type='WARNING';
              else if index(upcase(message),'UNINITIALIZED') then
type='UNINITIALIZED';
              else if index(upcase(message),'MERGE') and index(message,'BY') and
              index(upcase(message),'REPEATS') then type='MERGE BY ISSUE';
              else if index(upcase(message),'CONVERTED TO NUMERIC') then
type='CONVERTED TO NUMERIC';
```

```
            else if index(upcase(message),'CONVERTED TO CHARACTER') then
type='CONVERTED TO CHARACTER';

            program="&program";

            if type ne '' then do;
                found_issue = 1;
                output;
            end;

            /* At end of file, if nothing was output */
            if eof and found_issue = 0 then do;
                type = 'INFO';
                message = 'No issues found';
                output;
            end;
        run;

        filename lg_file clear;
  %end;

  data log_messages;
        set _xlog_messages:;
  run;

  /* clean up datasets */
  proc datasets lib=work nolist;
        delete _xlog_messages:;
  quit;

  /* Create summary counts by file and message type */
  proc sql;
        create table log_summary as
              select program, type, count(*) as message_count
              from log_messages group by program, type;
  quit;

  /* Generate HTML report */
  ods listing close;
  ods html path="&seefolder" file="log_summary.html" style=htmlblue;
  title1 "SAS Log Summary Report";
  title2 "&sysdate9 &systime";

  proc report data=log_summary nowd;
        column program type message_count;
        define program / "Log File";
        define type / "Message Type";
        define message_count / "Number of Messages";
  run;

  title "Detailed Log Messages";

  proc report data=log_messages nowd;
        column program type message;
        define program / "Log File";
        define type / "Message Type";
        define message / "Log Message" style(column)=[cellwidth=6in];
  run;

  ods html close;
  ods listing;

  /* clean up datasets */
  proc datasets lib=work nolist;
        delete log_list filelist log_messages log_summary;
  quit;
```

```
%mend;

%saseg_logcheck_all (dir=<your path>);
```