PharmaSUG 2026 - Paper AP-267

The Log Whisperer: Stop Reading SAS Logs. Start Ranking Them

Charu Shankar, SAS Institute Inc.

ABSTRACT

SAS logs are central to quality control, yet review remains slow, manual, and inconsistent. Programmers scan line by line, triage varies by reviewer, and valuable time is lost separating signal from noise.

This paper introduces a SAS-only "Log Whisperer" that transforms log review into a repeatable, data-driven workflow. A lightweight rules table defines what matters. A DATA step engine uses PRX pattern matching to detect issues at scale. PROC SQL then scores, ranks, and summarizes logs into a prioritized review queue.

Rather than reading everything, reviewers focus immediately on the highest-risk logs and the most common root causes.

Takeaway: a reusable SAS utility that reduces QC thrash, improves consistency, and turns log review into a structured, scalable process.

INTRODUCTION

Every SAS programmer reviews logs—few do it efficiently.

The typical workflow is simple: open the log, scroll, scan for ERROR or WARNING, and decide what matters. This approach is time-consuming, inconsistent, and difficult to scale. The issue is not effort—it is lack of structure.

This paper reframes log review from a reading task into a scoring system by treating logs as data. Once structured, logs can be parsed, scored, ranked, and summarized.

This shifts the question from:
"What does this log say?"
to
"Which logs matter most?"

THE WHISPERER MODEL

The solution reduces to five steps:

Step	Purpose	Outcome
1 DEFINE	What matters	Consistency
2 DETECT	Find patterns	Precision
3 SCORE	Measure impact	Signal
4 RANK	Prioritize logs	Action
5 INSIGHT	Root causes	Spotlight

👉 Key insight: Structure replaces subjectivity

STEP 1 — DEFINE: RULES

This rules table defines what to detect, how severe it is, and how much weight it contributes to the final score.

```
/*=====================================================================
STEP 1- DEFINE |Create rules table to define what & how much it matters
=====================================================================*/
data work.rules;
  length rule_id $10 severity_label $10 label $80 pattern $200;
  infile datalines dlm='~' dsd truncover;
  input rule_id :$10.
        severity :8.
        weight :8.
        severity_label :$10.
        label :$80.
        pattern :$200.;
datalines;

E001~4~50~CRITICAL~ERROR:~/^ERROR:/
W001~2~8~MINOR~WARNING:~/^WARNING:/
M001~3~15~MAJOR~Uninitialized~/^NOTE: .*uninitialized/i
M002~3~12~MAJOR~Invalid argument~/^NOTE: .*invalid argument/i
M003~3~10~MAJOR~Missing generated~/^NOTE: Missing values were
generated/i
;
run;
```

OUTPUT

RULES

	rule_id	severity_label	label	pattern	severity	weight
1	E001	CRITICAL	ERROR:	/(^	4	50
2	W001	MINOR	WARNING:	/(^	2	8
3	M001	MAJOR	Uninitialized	/uninitialized/i	3	15
4	M002	MAJOR	Invalid argument	/invalid argument/i	3	12
5	M003	MAJOR	Missing generated	/Missing values were generated/i	3	10

👉 **Key insight:** Rules standardize what matters.

STEP 2 — DETECT: DISCOVER LOG FILES

This step builds a dataset of all .log files in a Windows folder.

```
/*=====================================================================
  STEP 2 — DETECT | Discover logs + compile rules + scan lines
=====================================================================*/
%let LOG_ROOT=C:\temp\saslogs;
/* 2A — Discover log files */
filename _dir pipe "dir /s /b ""&LOG_ROOT.\*.log""";
```

```
data work.log_files;
  infile _dir truncover;
  input filepath $char500.;
 filename = scan(filepath,-1,'\');
  folder   = substr(filepath,1,length(filepath)-length(filename)-1);
run;
filename _dir clear;

/* 2B — Compile rules */

data work.rules_compiled;
  set work.rules;
  if missing(rule_id) or
missing(pattern) then delete;
  prx_id = prxparse(pattern);
  if missing(prx_id) then do;
    putlog "ERROR: Invalid regex pattern: " pattern=;
    stop;
  end;
run;

/* 2C — Scan logs */

data work.findings;
  length filepath folder filename thisfile $500
         line $32767
         rule_id $10
         severity_label $10
         label $80
         pattern $200
         severity 8
         weight 8
         i 8
         nobs_rules 8;

  set work.log_files;
  thisfile = filepath;

  infile dummy filevar=thisfile end=eof lrecl=32767 truncover;

  do while (not eof);
    input line $char32767.;

    do i = 1 to nobs_rules;
      set work.rules
          (keep=rule_id severity weight severity_label label pattern)
          point=i nobs=nobs_rules;

      if not missing(pattern) then do;
        if prxmatch(pattern, line) then output;
      end;
    end;
  end;
  stop;
run;
```

👉 **Key insight:** Detection becomes consistent and scalable. Each regex is compiled once and stored for reuse.

STEP 3 — SCORE: SUMMARIZE FINDINGS TO FILE LEVEL

Line-level findings are rolled up to file-level scores.

```
/*=====================================================================
STEP 3 — SCORE  Summarize line-level findings into file-level scores
=====================================================================*/
proc sql;
  create table work.log_summary as
  select
    f.folder,
    f.filename,
    f.filepath,
    coalesce(sum(a.weight),0) as score,
    sum(case when a.severity=4 then 1 else 0 end) as n_critical,
    sum(case when a.severity=3 then 1 else 0 end) as n_major,
    sum(case when a.severity=2 then 1 else 0 end) as n_minor,
    sum(case when a.severity=1 then 1 else 0 end) as n_info,
    count(a.rule_id) as n_hits
  from work.log_files f
  left join work.findings a
    on f.filepath=a.filepath
  group by f.folder, f.filename, f.filepath
  order by score desc, n_critical desc, n_major desc, n_hits desc;
quit;
```

👉 **Key insight:** Not all issues are equal.

STEP 4 — RANK: ADD REVIEW STATUS

This adds an executive-friendly review signal.

```
/*=====================================================================
STEP 4— RANK | Add review status
=====================================================================*/
data work.log_summary;
  set work.log_summary;
  length status $8;

  if n_critical > 0 then status='RED';
  else if n_major > 0 then status='AMBER';
  else if n_minor > 0 then status='YELLOW';
  else status='GREEN';
run;
```

The output is no longer a log—it is a ranked review queue.

SAMPLE OUTPUT: TOP 5 LOGS

Rank	Log File	Score	Critical	Major	Hits	Status
1	ae_run.log	120	2	3	15	RED
2	dm_build.log	75	1	2	10	AMBER
3	vs_check.log	40	0	2	8	YELLOW

Rank	Log File	Score	Critical	Major	Hits	Status
4	lb_clean.log	15	0	1	3	YELLOW
5	ex_merge.log	0	0	0	0	GREEN

👉 Key insight: Review order becomes obvious.

STEP 5 — INSIGHT: ROOT CAUSE FREQUENCY

This identifies the most frequent recurring issues across all logs.

```
/*===================================
STEP 5 - INSIGHT | Root cause frequency
===================================*/;
proc sql;
  create table work.rule_frequency as
  select
    rule_id,
    severity_label,
    label,
    count(*) as hits
  from work.findings
  group by rule_id, severity_label, label
  order by hits desc;
quit;
```

👉 Key insight: Repeated issues reveal where the process breaks down.

WHAT MAKES THIS DIFFERENT

Most approaches detect issues. This approach prioritizes them, quantifies them and explains them. This is the difference between debugging and managing risk.

CONCLUSION

Manual log review does not scale.
By treating logs as data, detection becomes automated, importance becomes measurable, review becomes prioritized. The result is faster, more consistent, and more meaningful QC.
Stop reading logs. Start ranking them.

REFERENCES

- David L. Cassell, “PRX Functions and CALL Routines”
- David L. Cassell, “The Basics of the PRX Functions”
- Philip Mason, “Analyzing Your SAS Log with User Defined Rules Using an App or Macro”
- Drew Metz, “SAS Log Parsing…”
- Richann Watson, “Check Please: An Automated Approach to Log Checking”
- Valerie Williams and Ganesh Prasad, “Automated Log Checking with LOG_CHECK.SAS”
- SAS Institute Inc., “PRXPARSE Function”
- SAS Institute Inc., “PRXMATCH Function”

CONTACT INFORMATION

Charu Shankar
SAS Institute Inc.
Charu.Shankar@sas.com