PharmaSUG 2024 - Paper DS- 109

Analyzing your SAS log with user defined rules using an app or macro.

Philip Mason, Wood Street Consultants Ltd.

ABSTRACT

SAS provide some pretty basic help with logs that are produced, typically just linking to errors and warnings. Many people build log checkers to look for particular things of interest in their logs, which usually involves capturing the log and then running some SAS code against it. I made a way to define rules in JSON format which can be read by a SAS macro and used to look for things in a log. This means different rules can be used for different use cases. They can be used via a macro or via a web application I build. The web app can switch between rules, provides summaries, draws diagrams of the code, provides performance stats, and more. Hopefully this functionality might one day be built into SAS, but in the meantime it works well as an addition.

INTRODUCTION

At many places I have worked there are log checkers developed in order to improve the checking of logs in various ways. Sometimes it is to automate the process in batch jobs that run regularly. Sometimes it is to be consistent and complete in the checking of logs, rather than relying on the expertise of individuals to do so. Some logs checkers only look for the main issues such as the things that SAS label as Errors and Warnings. But other times we might have specific things to check for when we are concerned about characters being automatically converted to numerics, etc. And other times we might want to look for custom messages we write to the log ourselves, in order to highlight problems in data or to summarize information.

Since there are so many use cases for log checking I thought it would be useful to create a tool that could be used to analyze logs while being able to use different sets of requirements/rules. Different rules that define what we want to find in a log would allow us to examine a log in different ways for different use cases.

Most log checkers that I have seen in the past have been SAS programs or macros that could be called to analyze a log and produce a report. I have also written some applications in the past to display a log along with some kind of interactive viewing and analysis like a count of how many errors and warnings are in it. I wanted something that would work both from a program and from a user interface.

At my current place of work we use both PC SAS and Life Science Application Framework (LSAF). Neither of these programming environments would do any more than highlight the log (only PC SAS) and provide links to Errors and warnings (only LSAF). I knew I could improve on that.

APPROACH

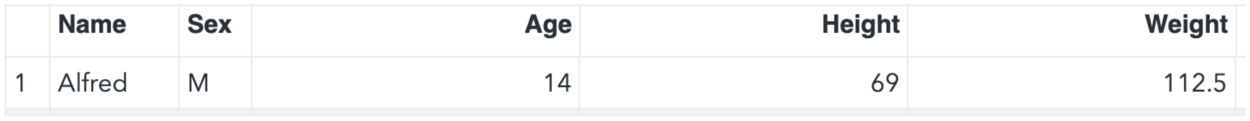
Based on previous log checkers I have written I decided that the primary way of checking a log would be a web application, since it can run in a browser on any computer and provides a huge amount of functionality. The web application needs to be able to load a SAS log in, which will vary depending on what variety of SAS is being used. In analyzing a log in a web application, I have access to all the text of the log and so can extend the analysis to display some summary information, such as what data was used, what macros were used, how long things took to run, etc.

I also wanted to create a SAS macro that could be used after a SAS program to check its log. Or that SAS macro could be called against a SAS log or logs that have been previously run and saved.

For the web application and SAS macro to analyze logs in the same way I needed a way to represent rules for the analysis in a form that could be used by both a JavaScript program and a SAS program. That would include specifying what to look for in each line of the log as well as how to format the line of the log when viewed. Additionally, I wanted to provide some summary information like a list of errors and warnings that could be clicked on and then jump to that location in the SAS log. That would provide not only a speedy way to find problems, but also then to be able to view those messages in their context within the SAS log.

DEFINING RULES

I decided to define rules for the log viewer in JSON format, which stands for JavaScript object notation. It is a great text format that is easily read by people and machines. It is well supported by SAS, JavaScript, and most other languages, which makes it a great choice since people can edit rules using their favorite method. JSON can represent tabular data in a simple way. Basically rows/observations of a table/dataset appear in square brackets, which indicate an array element. Within each row you would have the value for each variable/column. So for instance if you take the first line in sashelp.class it looks like this



	Name	Sex	Age	Height	Weight
1	Alfred	M	14	69	112.5

But if that was represented in JSON format, it would look like this

```
{"Name":"Alfred","Sex":"M","Age":14,"Height":69,"Weight":112.5}
```

In thinking about what data I need to define a rule for the log viewer the simplest might be to find lines that start with NOTE:, which I would then want to color blue in the log I display so that they look like NOTE messages in SAS. So to implement that I will have a startsWith key, that could have a value of NOTES:. In JSON that would look like this:

```
"startswith": "NOTE:"
```

Then I want to display that in blue. To do so it seemed like the most flexible way would be to use some HTML styling, in which case I would need to put some HTML in front of the line and finish the line with some HTML, since HTML always has open and close tags around text that it operates on. I can use CSS styles (Cascading Style Sheets) in the HTML with the style keyword. So, to make a line blue I decided to add these bits to the rule:

```
"startswith": "NOTE:",
"prefix": "<span style='color: blue'>",
"suffix": "</span>"
```

Next, I was thinking about errors and warnings that I would find in a log that I would want the user to be able to jump to in the log, so they could be quickly found in thousands of lines. That would need some things to appear in a list of links and others to not appear – since I don't want to link to every note in a log for instance. And I also want to specify a color that the link will be displayed in, so the rule became:

```
  "startswith": "NOTE:",
  "prefix": "<span style='color: blue'>",
  "suffix": "</span>",
  "anchor": false,
  "linkColor": "blue",
```

Next consideration was that I wanted to look for different things but categorize them so that if there were lots of messages, I could choose to just look at specific categories. For instance, I might look for 12 different error messages, but would want them all to be categorized as errors. So, my rule gained another attribute:

```
  "startswith": "NOTE:",
  "prefix": "<span style='color: blue'>",
  "suffix": "</span>",
  "anchor": false,
  "linkColor": "blue",
  "type": "NOTE",
```

Next, I realized that some things I look for are not at the start of a line, but somewhere in the middle or arranged in some kind of pattern. So, I decided I would add another way to find things and that would be using regular expressions, since that should be flexible enough to let me find almost anything. So, I added a rule type key, and the warning messages I look for would use a rule like this:

```
    "ruleType": "regex",
    "regex": "(?:^WARNING:|^WARNING\\s)(?!.*PRODUCT WITH WHICH|.*PRODUCT WITH WHICH|.*IS ASSOCIATED WILL EXPIRE|.*MPRINT|.*MLOGIC|.*SYMBOLGEN|.*IS ASSOCIATED IS SCHEDULED TO EXPIRE|.*YOUR SYSTEM IS SCHEDULED TO EXPIRE)",
    "prefix": "<span style='color: green'>",
    "suffix": "</span>",
    "anchor": true,
    "linkColor": "green",
    "type": "WARN",
```

I added an item to allow an optional description for the rule, to make things clearer to others who might be looking at them.

Another minor item I added to the rules was an attribute to indicate if something was **interesting** or not, which would mean I could look through the log checking each thing that I found interesting which would give me flexibility to decide what I wanted to look at.

And a final item I added was something to allow turning lines I found in the log into links to take me to other web sites or applications, etc. It made a lot of powerful and flexible customizations possible. The following rule finds text that looks like a path to a directory in LSAF and turns that into a link which opens my fileviewer application to show the contents of that directory. The text that the regular expression matches is substituted into the HTML or prefix and suffix where is has the placeholder **{{matched}}**:

```
    "description": "Look for paths to a file or directory starting with /general. Then create a link to fileviewer.",
    "ruleType": "regex",
    "regex": "\\/general[\\/\\w+|\\-|.]*",
    "prefix": "<a href='https://xarprod.ondemand.sas.com/lsaf/filedownload/sdd:/general/biostat/tools/fileviewer/index.html?file={{matched}}' target='_blank'>",
    "suffix": "{{matched}}</a>",
    "anchor": false,
    "linkColor": "blue",
    "type": "FILE",
    "interesting": false,
    "substitute": true
```

An example of a rules file can be found here: https://github.com/philipmason/Wood-Street-Consultants/blob/main/Log%20Analysis/rules.json

USING RULES WITH A MACRO

Now that we have rules to define what to look for, we need things to use those rules. Firstly, a macro is the simplest of these. A lot of the information provided in the rules is not needed, as we are not displaying it in an interactive form. The aim is to do the analysis and produce a report of what was found in the log. But I also want to create some kind of macro variable that would have an overall result. For instance, if we wanted to check for errors, we should be ablet to specify to use some rules, but just check for errors and provide a macro variable that could be checked to see if any were found or if the log was error free. Being able to specify what to look for means that we don't need to check all the rules each time but can use the type to select the things we want to look for.

I created a macro called fileCheckWithRules, since this can be used against any text file and doesn't just have to be a SAS log. The parameters are:

- **log** - First positional parameter is the path to the SAS log
- **rules** – path to the rules JSON file
- **returnTypes** – list of types you want returned matching types from the rules JSON file. They should be separated by | characters, e.g. returntypes=ERROR|WARN|SERIOUS
- **outputName** – name of table to create with the results of log analysis
- pri|nt – 1 would create a report, 0 doesn't create a report.

The macro creates a global macro variable that matches the outputName (e.g. filecheckwithrules) which contains the number of rows in the results of the log analysis output dataset. That means that if you specified returnTypes=ERROR then the output data would only have errors, and the macro variable would have the number of errors found. If you use the default of returnTypes=ERROR|WARN|SERIOUS then the macro variable has the number of rows in the output dataset, meaning that if it was 0 then there were no errors, warnings or serious things found. Changing the value of outputName lets you choose a different name for both the dataset and the macro variable.

Code for this can be found here: https://github.com/philipmason/Wood-Street-Consultants/blob/main/Log%20Analysis/filecheckwithrules.sas

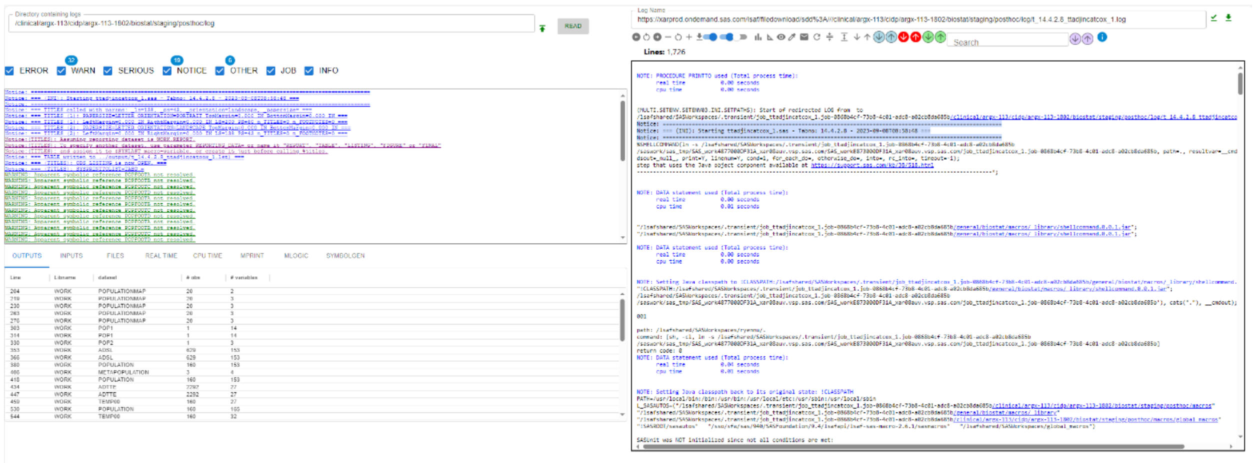
USING RULES WITH A WEB APPLICATION

The macro described in the previous section uses SAS code to make use of the rules and analyse a SAS log, providing an output dataset and macro variable. This section extends the use of the rules to provide not only an analysis of the log and summary, but a huge amount of extra detail and help. It is provided by using a web application, which is a JavaScript application that runs in a web browser such as Microsoft Edge, meaning that it can be used on computers/tablets/phones. For those web developers out there, it is written using the following technology: ES6, CSS 3, HTML 5, React 18, React Router, MUI, MUI Data Grid, React Select, Mermaid. The JavaScript libraries used are available via https://www.npmjs.com/.

I called this application the log viewer since it provides an interface allowing viewing of a SAS log that is highlighted according to rules defined in the JSON rules file. It was created to allow users to view a log in a better way while also improving and standardizing the analysis of SAS logs.

When using the log viewer, a log is selected and loaded by the JavaScript code into an array. A set of rules is also loaded from the JSON file into an array. It is then possible to step through each line of the log and each rule to check everything. Lines can be transformed according to the rules used for display in a window. Things found are tracked by updating other arrays so we know what we have found, whether it should be displayed in a list of links, how things should be colored, what line in a log should be linked to, and so on.

The log viewer looks like this:



Directory containing logs
/clinical/argx-113/cidp/argx-113-1802/biostat/staging/posthoc/log

READ

32 19 0
ERROR WARN SERIOUS NOTICE OTHER JOB INFO

OUTPUTS INPUTS FILES REAL TIME CPU TIME MPRINT MLOGIC SYMBOLGEN

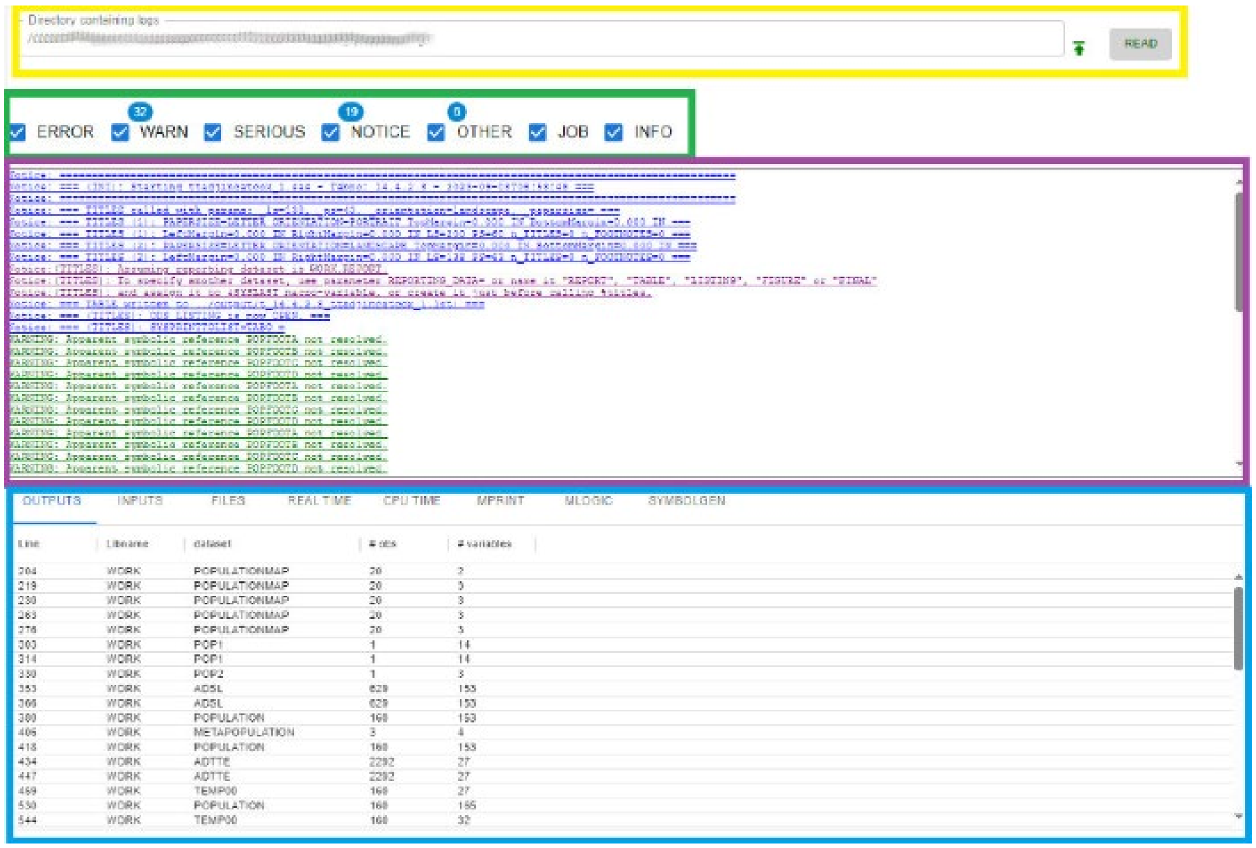
Log Name
https://xarprod.ondemand.sas.com/lsaf/filedownload/sdd%3A///clinical/argx-113/cidp/argx-113-1802/biostat/staging/posthoc/log/t_14.4.2.8_ttadjincatcox_1.log

Search

Lines: 1,726

```
NOTE: PROCEDURE PRINTTO used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds


(MULTI.SETENV.SETENV03.INI.SETPATHS): Start of redirected LOG from  to
/lsafshared/SASWorkspaces/.transient/job_ttadjincatcox_1.job-0868b4cf-73b8-4c01-adc8-a02cb8da685b/clinical/argx-113/cidp/argx-113-1802/biostat/staging/posthoc/log/t_14.4.2.8_ttadjincatco
Notice: ============================================================================================
Notice: === (INI): Starting ttadjincatcox_1.sas - Tabno: 14.4.2.8 - 2023-09-08T08:58:48 ===
Notice: ============================================================================================
%SHELLCOMMAND(ln -s /lsafshared/SASWorkspaces/.transient/job_ttadjincatcox_1.job-0868b4cf-73b8-4c01-adc8-a02cb8da685b
/saswork/sas_tmp/SAS_work4877008DF31A_xar08auv.vsp.sas.com/SAS_workE873000DF31A_xar08auv.vsp.sas.com/job_ttadjincatcox_1.job-0868b4cf-73b8-4c01-adc8-a02cb8da685b, path=., resultvar=__cmd
dsout=_null_, print=Y, linenum=Y, cond=1, for_each_do=, otherwise_do=, into=, rc_into=, timeout=-1);
step that uses the Java object component available at https://support.sas.com/kb/38/518.html
-------------------------------------------------------------------------------------------*/;


NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.01 seconds


"/lsafshared/SASWorkspaces/.transient/job_ttadjincatcox_1.job-0868b4cf-73b8-4c01-adc8-a02cb8da685b/general/biostat/macros/_library/shellcommand.0.0.1.jar";
"/lsafshared/SASWorkspaces/.transient/job_ttadjincatcox_1.job-0868b4cf-73b8-4c01-adc8-a02cb8da685b/general/biostat/macros/_library/shellcommand.0.0.1.jar";

NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds


NOTE: Setting Java classpath to !CLASSPATH:/lsafshared/SASWorkspaces/.transient/job_ttadjincatcox_1.job-0868b4cf-73b8-4c01-adc8-a02cb8da685b/general/biostat/macros/_library/shellcommand.
"!CLASSPATH:/lsafshared/SASWorkspaces/.transient/job_ttadjincatcox_1.job-0868b4cf-73b8-4c01-adc8-a02cb8da685b/general/biostat/macros/_library/shellcommand.0.0.1.jar";
/lsafshared/SASWorkspaces/.transient/job_ttadjincatcox_1.job-0868b4cf-73b8-4c01-adc8-a02cb8da685b
/saswork/sas_tmp/SAS_work4877008DF31A_xar08auv.vsp.sas.com/SAS_workE873000DF31A_xar08auv.vsp.sas.com/job_ttadjincatcox_1.job-0868b4cf-73b8-4c01-adc8-a02cb8da685b'), cats("."), __cmdout);

001

path: /lsafshared/SASWorkspaces/ryennu/.
command: [sh, -cl, ln -s /lsafshared/SASWorkspaces/.transient/job_ttadjincatcox_1.job-0868b4cf-73b8-4c01-adc8-a02cb8da685b
/saswork/sas_tmp/SAS_work4877008DF31A_xar08auv.vsp.sas.com/SAS_workE873000DF31A_xar08auv.vsp.sas.com/job_ttadjincatcox_1.job-0868b4cf-73b8-4c01-adc8-a02cb8da685b]
return code: 0
NOTE: DATA statement used (Total process time):
      real time           0.04 seconds
      cpu time            0.01 seconds


NOTE: Setting Java classpath back to its original state: !CLASSPATH
PATH=/usr/local/bin:/bin:/usr/bin:/usr/local/etc:/usr/sbin:/usr/local/sbin
L_SASAUTOS=("/lsafshared/SASWorkspaces/.transient/job_ttadjincatcox_1.job-0868b4cf-73b8-4c01-adc8-a02cb8da685b/clinical/argx-113/cidp/argx-113-1802/biostat/staging/posthoc/macros"
"/lsafshared/SASWorkspaces/.transient/job_ttadjincatcox_1.job-0868b4cf-73b8-4c01-adc8-a02cb8da685b/general/biostat/macros/_library"
"/lsafshared/SASWorkspaces/.transient/job_ttadjincatcox_1.job-0868b4cf-73b8-4c01-adc8-a02cb8da685b/clinical/argx-113/cidp/argx-113-1802/biostat/staging/posthoc/macros/global_macros"
"!SASROOT/sasautos"   "/sso/sfw/sas/940/SASFoundation/9.4/lsafapi/lsaf-sas-macro-2.6.1/sasmacros"   "/lsafshared/SASWorkspaces/global_macros")

SASUnit was NOT initialized since not all conditions are met:
```

There are two halves of the log viewer screen. The **left side** of screen looks like this:



Directory containing logs
[illegible]

READ

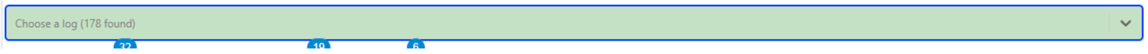
32 19 0
ERROR WARN SERIOUS NOTICE OTHER JOB INFO

```
Notice: ============================================================================================
Notice: === (INI): Starting ttadjincatcox_1.sas - Tabno: 14.4.2.8 - 2023-09-08T08:58:48 ===
Notice: ============================================================================================
Notice: === TITLES called with params: _ls=131, _ps=43, _orientation=landscape, _papersize= ===
Notice: === TITLES (1): PAPERSIZE=LETTER ORIENTATION=PORTRAIT TopMargin=0.300 IN BottomMargin=0.080 IN ===
Notice: === TITLES (1): LeftMargin=0.300 IN RightMargin=0.300 IN LS=100 PS=60 n_TITLES=1 n_FOOTNOTES=0 ===
Notice: === TITLES (2): PAPERSIZE=LETTER ORIENTATION=LANDSCAPE TopMargin=0.000 IN BottomMargin=0.000 IN ===
Notice: === TITLES (2): LeftMargin=0.000 IN RightMargin=0.000 IN LS=131 PS=43 n_TITLES=0 n_FOOTNOTES=0 ===
Notice:(TITLES): Assuming reporting dataset is WORK.REPORT.
Notice:(TITLES): To specify another dataset, use parameter REPORTING_DATA= or name it "REPORT", "TABLE", "LISTING", "FIGURE" or "FINAL"
Notice:(TITLES): and assign it to &SYSLAST macro-variable, or create it just before calling titles.
Notice: === TABLE written to ../output/t_14.4.2.8_ttadjincatcox_1.lst ===
Notice: === (TITLES): ODS LISTING is now OPEN. ===
Notice: === (TITLES): SYSPRINTTOLIST=CARD =
WARNING: Apparent symbolic reference BOPFOOTA not resolved.
WARNING: Apparent symbolic reference BOPFOOTB not resolved.
WARNING: Apparent symbolic reference BOPFOOTC not resolved.
WARNING: Apparent symbolic reference BOPFOOTD not resolved.
WARNING: Apparent symbolic reference BOPFOOTA not resolved.
WARNING: Apparent symbolic reference BOPFOOTB not resolved.
WARNING: Apparent symbolic reference BOPFOOTC not resolved.
WARNING: Apparent symbolic reference BOPFOOTD not resolved.
WARNING: Apparent symbolic reference BOPFOOTA not resolved.
WARNING: Apparent symbolic reference BOPFOOTB not resolved.
WARNING: Apparent symbolic reference BOPFOOTC not resolved.
WARNING: Apparent symbolic reference BOPFOOTD not resolved.
```

OUTPUTS INPUTS FILES REAL TIME CPU TIME MPRINT MLOGIC SYMBOLGEN

Line	Libname	dataset	# obs	# variables
204	WORK	POPULATIONMAP	20	2
219	WORK	POPULATIONMAP	20	3
230	WORK	POPULATIONMAP	20	3
263	WORK	POPULATIONMAP	20	3
276	WORK	POPULATIONMAP	20	3
303	WORK	POP1	1	14
314	WORK	POP1	1	14
330	WORK	POP2	1	3
353	WORK	ADSL	629	153
366	WORK	ADSL	629	153
380	WORK	POPULATION	160	153
406	WORK	METAPOPULATION	3	4
418	WORK	POPULATION	160	153
434	WORK	ADTTE	2292	27
447	WORK	ADTTE	2292	27
469	WORK	TEMP00	160	27
530	WORK	POPULATION	160	155
544	WORK	TEMP00	160	32

- The top section shows the directory for the current log being viewed.
  - Pressing the READ button will read that directory and show a selector with all the logs available. You can then select one of those logs to load that in and analyze it.

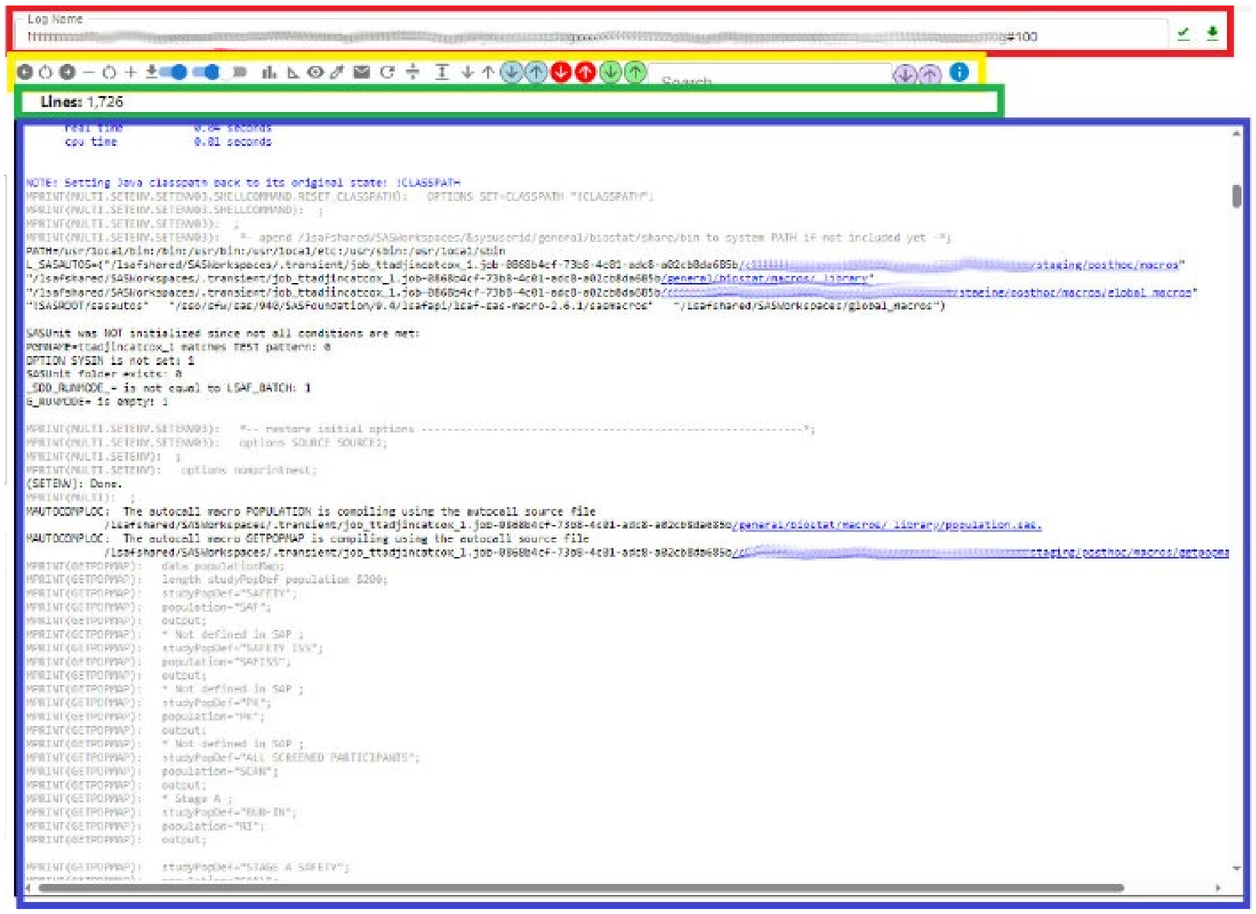


Choose a log (178 found)

  - Pressing the icon will read the parent directory of the current directory, and display any logs found there in a selector.

- The next section shows a list of categories of things we were looking for. This list will vary depending on what rules JSON file we are using. If a number is shown next to a category, then that indicates how many lines were found for that.
    - Check boxes can be clicked to determine what is shown in the area below. This is useful especially when there are lots of lines and you want to focus on particular categories.
- The middle section displays a list of links for the categories selected above.
    - Clicking on a link will jump to that position in the log.
    - Colors are determined in the rules JSON file being used.
- The bottom section has 8 tabs each of which displays a table. The first 5 tables allow clicking on a row and jumping to the corresponding position in the log.
    - Outputs – output datasets taken from messages at the end of each step.
    - Inputs – input datasets taken from messages at the end of each step.
    - Files – external files that are either read or written.
    - Real Time – real time taken from messages at the end of each step.
    - Cpu Time – cpu time taken from messages at the end of each step.
    - Mprint – Shows how many lines were found for macros in Mprint log messages. Checkboxes can be used to turn off those mprint messages in the log (useful if there are many).
    - Mlogic – as for mprint but for Mlogic.
    - Symbolgen – as for mprint but for symbolgen.

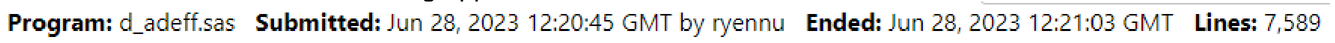
The **right side** of the screen looks like this:



```
      real time           0.84 seconds
      cpu time            0.81 seconds


NOTE: Setting Java classpath back to its original state: !CLASSPATH
MPRINT(MULTI.SETENV.SETENV03.SHELLCOMMAND.RESET_CLASSPATH):   OPTIONS SET=CLASSPATH "!CLASSPATH";
MPRINT(MULTI.SETENV.SETENV03.SHELLCOMMAND):   ;
MPRINT(MULTI.SETENV.SETENV03):   ;
MPRINT(MULTI.SETENV.SETENV03):   *- append /lsafshared/SASWorkspaces/&sysuserid/general/biostat/share/bin to system PATH if not included yet -*;
PATH=/usr/local/bin:/bin:/usr/bin:/usr/local/etc:/usr/sbin:/usr/local/sbin
L_SASAUTOS=("/lsafshared/SASWorkspaces/.transient/job_ttadjincatcox_1.job-0868b4cf-73b8-4c01-adc8-a02cb8da685b/[redacted]/staging/posthoc/macros"
"/lsafshared/SASWorkspaces/.transient/job_ttadjincatcox_1.job-0868b4cf-73b8-4c01-adc8-a02cb8da685b/general/biostat/macros/_library"
"/lsafshared/SASWorkspaces/.transient/job_ttadjincatcox_1.job-0868b4cf-73b8-4c01-adc8-a02cb8da685b/[redacted]/staging/posthoc/macros/global_macros"
"!SASROOT/sasautos"   "/sso/sfw/sas/940/SASFoundation/9.4/lsafapi/lsaf-sas-macro-2.6.1/sasmacros"   "/lsafshared/SASWorkspaces/global_macros")

SASUnit was NOT initialized since not all conditions are met:
PGMNAME=ttadjincatcox_1 matches TEST pattern: 0
OPTION SYSIN is not set: 1
SASUnit folder exists: 0
_SDD_RUNMODE_= is not equal to LSAF_BATCH: 1
G_RUNMODE= is empty! 1

MPRINT(MULTI.SETENV.SETENV03):   *-- restore initial options ------------------------------------------------------------*;
MPRINT(MULTI.SETENV.SETENV03):   options SOURCE SOURCE2;
MPRINT(MULTI.SETENV):   ;
MPRINT(MULTI.SETENV):   options nomprintnest;
(SETENV): Done.
MPRINT(MULTI):   ;
MAUTOCOMPLOC:  The autocall macro POPULATION is compiling using the autocall source file
          /lsafshared/SASWorkspaces/.transient/job_ttadjincatcox_1.job-0868b4cf-73b8-4c01-adc8-a02cb8da685b/general/biostat/macros/_library/population.sas.
MAUTOCOMPLOC:  The autocall macro GETPOPMAP is compiling using the autocall source file
          /lsafshared/SASWorkspaces/.transient/job_ttadjincatcox_1.job-0868b4cf-73b8-4c01-adc8-a02cb8da685b/[redacted]/staging/posthoc/macros/getpopma
MPRINT(GETPOPMAP):   data populationMap;
MPRINT(GETPOPMAP):   length studyPopDef population $200;
MPRINT(GETPOPMAP):   studyPopDef="SAFETY";
MPRINT(GETPOPMAP):   population="SAF";
MPRINT(GETPOPMAP):   output;
MPRINT(GETPOPMAP):   * Not defined in SAP ;
MPRINT(GETPOPMAP):   studyPopDef="SAFETY ISS";
MPRINT(GETPOPMAP):   population="SAFISS";
MPRINT(GETPOPMAP):   output;
MPRINT(GETPOPMAP):   * Not defined in SAP ;
MPRINT(GETPOPMAP):   studyPopDef="PK";
MPRINT(GETPOPMAP):   population="PK";
MPRINT(GETPOPMAP):   output;
MPRINT(GETPOPMAP):   * Not defined in SAP ;
MPRINT(GETPOPMAP):   studyPopDef="ALL SCREENED PARTICIPANTS";
MPRINT(GETPOPMAP):   population="SCAN";
MPRINT(GETPOPMAP):   output;
MPRINT(GETPOPMAP):   * Stage A ;
MPRINT(GETPOPMAP):   studyPopDef="RUN-IN";
MPRINT(GETPOPMAP):   population="RI";
MPRINT(GETPOPMAP):   output;

MPRINT(GETPOPMAP):   studyPopDef="STAGE A SAFETY";
```

- The top section shows the name of the log file currently loaded.
    - You can enter a log name into this field and press ✓ to load it.
    - You will be able to click on ⤓ in order to display a list of versions of this log. This is not yet implemented but is coming soon.
- The toolbar is next and has many tools for interacting with the log analysis.
    - Adjusts the width of the left and right sides of the screen. You can move the center divider to the left or right or return it to 50:50.
    - − ⟲ + Change the font size of text on the screen making it smaller or bigger, or returning it to the default of 12 point.
    - ⤓ Download the log file.
    -
        - Show/Hide source lines, which is very useful to reduce the clutter around more important parts of the log.
        - Show/Hide macro logging lines such as Mprint, Mlogic, Symbolgen, Mautocomploc, which is also very useful in removing clutter to let you focus on more important parts. You can always switch this back on to display them again.

        - Show/Hide line numbers – this is useful as it turns line numbers on in the log area as well as for the links in the left side. It helps to identify if there are a lot of messages near each other when displayed in the list of links.
    - Paste the contents of the clipboard in as a new log to be analyzed. This means you can grab a log from anywhere and paste it in to use with the log viewer.
    - Displays a **diagram** showing the data flow based on info from the SAS log. I take messages at the end of steps which say what data was used as input and what was output, to generate a dot format diagram that can be rendered by libraries such as Mermaid (which we use).
    - Choose a JSON file containing **rules** to be used in the log analysis. Several JSON files are provided when the application is built, but it is possible for people to build more that can be added to the list.
    - Displays the **current rules** in a table. Useful to see what rules are in effect right now.
    - Extracts the **source code** from the log (if available) and puts it into the clipboard. You can then go and paste it into another editor to see or use it.
    - Prepare an **email** to be sent with a link to this log. Useful if you want to notify someone about something to do with the log.
    - **Refresh** the analysis and screen.
    - Make the log viewing area a little bit **smaller** or **larger**.
    - Go **down** or **up** one page.
    - Find the next or previous **interesting thing** in the log. Interesting things are defined in the rules JSON file.
    - Find the next or previous **error** in the log. They are defined in the rules JSON file, so may not correspond exactly to what SAS considers an error.
    - Find the next or previous **warning** in the log. They are defined in the rules JSON file, so may not correspond exactly to what SAS considers a warning.
    - Search Enter text to **search** for in the log, and then find the next or previous match.
    - Display an information popup with info about this screen and how to use it.
- Some basic information about the log appears next.

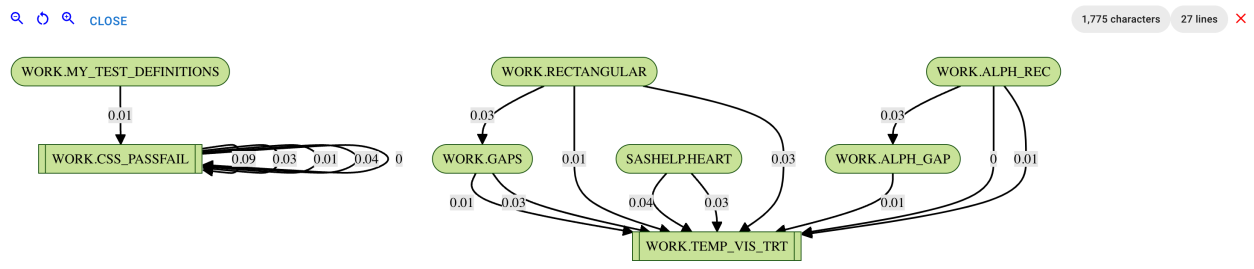

Program: d_adeff.sas **Submitted:** Jun 28, 2023 12:20:45 GMT by ryennu **Ended:** Jun 28, 2023 12:21:03 GMT **Lines:** 7,589

    - Lines of text in the log in which we will always get the number of lines.
    - For some jobs we get further details taken from headers we use.
- The main area displays the content of the log.
    - We remove the page headers from the SAS log (e.g. page number).
    - We try to identify text that looks like a path in LSAF, and we turn that into a hyperlink so that you can click on it and link to the file viewer web application (which is an app that shows files/directories in the web browser), which will show you the files in that directory and let you view them (if possible).

- Lines of the log are displayed using the rules, which can add HTML before and after each line. This means color, font, links and much more can be specified for lines that match a rule. e.g. NOTE: lines appear in blue.

EXAMPLE DATA FLOW DIAGRAM

These can be automatically produced by the log viewer. Here is an example.



AUTOMATING USE OF LOG VIEWER

You can open the log viewer and load logs in to be processed. However, if using SAS from a some environments I have created an automated way to open the log viewer from within SAS.

<u>LSAF</u>

However, when using from LSAF I have setup a bookmarklet which will identify where you are in your navigation of the file system and automatically invoke the log viewer displaying the selected directory or log. A bookmarklet is a JavaScript program which can run from a bookmark in your browser. They are very flexible and powerful, but beyond the scope of this paper.

<u>PC SAS</u>

I setup a tool in the toolbar which allocates a fileref with the clipbrd engine, it then saves the contents of the log to the clipboard, then starts the log viewer web application passing a parameter to cause it to paste the clipboard in. This means that the PC SAS user can click on the log viewer tool in the toolbar and open that log in the log viewer.

<u>SAS Studio</u>

I'm working on a bookmarklet to automate this, but in the meantime, you can use the tool to paste a log from the clipboard.

<u>VS Code</u>

I am planning to make an extension for VS code to make the log viewer available in VS Code, so we can press a key combination and open the SAS log in the log viewer. But in the meantime, you can copy the log to the clipboard and paste it into the log viewer.

BENEFITS OF THE LOG VIEWER

The log viewer provides many benefits including:

- Fast and interactive, allows switching between different rules.
- Finds everything you are interested in very quickly, providing counts by category and links to them in the log.
- Automatic Data flow diagram generation.
- Switch lines in log on/off – source lines, mprint lines
- Extract source code from log.
- Generate email with link to log being viewed, so others can view the same log.

- Step through log by page, error, warning, interesting thing.
- Provides summary tables showing data used, and linking to each location in the log.
- Helps with performance analysis showing summary tables with real & CPU time used in steps
- Can be used for non-SAS logs, or any text file analysis

CONCLUSIONS

The log viewer provides a great tool for the SAS programmer, no matter what platform you use SAS on. You can define rules to find what you are interested in and display it with the style you want.

You can use the same rules to analyze logs from SAS programs run in batch.

Using the same rules for both interactive and batch analysis means your results are consistent and high quality. Analysis is done quickly.

CONTACT INFORMATION

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

Philip Mason
Wood Street Consultants Ltd.
phil@woodstreet.org.uk
https://github.com/philipmason/Wood-Street-Consultants