PharmaSUG 2014 - Paper AD12

MACUMBA - The log belongs to the program

Michael Weiss, Bayer Pharma AG, Berlin, Germany

ABSTRACT

MACUMBA is an in-house-developed application for SAS® programming. It combines interactive development features of PC-SAS, the possibility of a client-server environment and unique state-of-the-art features. This presentation shows a feature of the MACUMBA application that aligns the generated log file to the SAS program, it belongs to. In this way all relevant log lines (ERROR's, WARNING's and others) are directly displayed at the source code line, they occurred. This alignment provides an easy way to find the source of a problem and ensures that no important log line is overlooked anymore.

INTRODUCTION

The MACUMBA development was started in 2007 as a graphical implementation of the PC-SAS DATA step debugger. Step by step (beside the daily work) lots of other features where added to support SAS program and macro development, data examinations and a lot more. Figure 1 shows a basic overview of the application design that is similar to common development environments of other programming languages.

MACUMBA is implemented in pure Java and uses the SAS IOM technology to access a SAS Object Spawner for "interactive like" SAS code execution and data access.

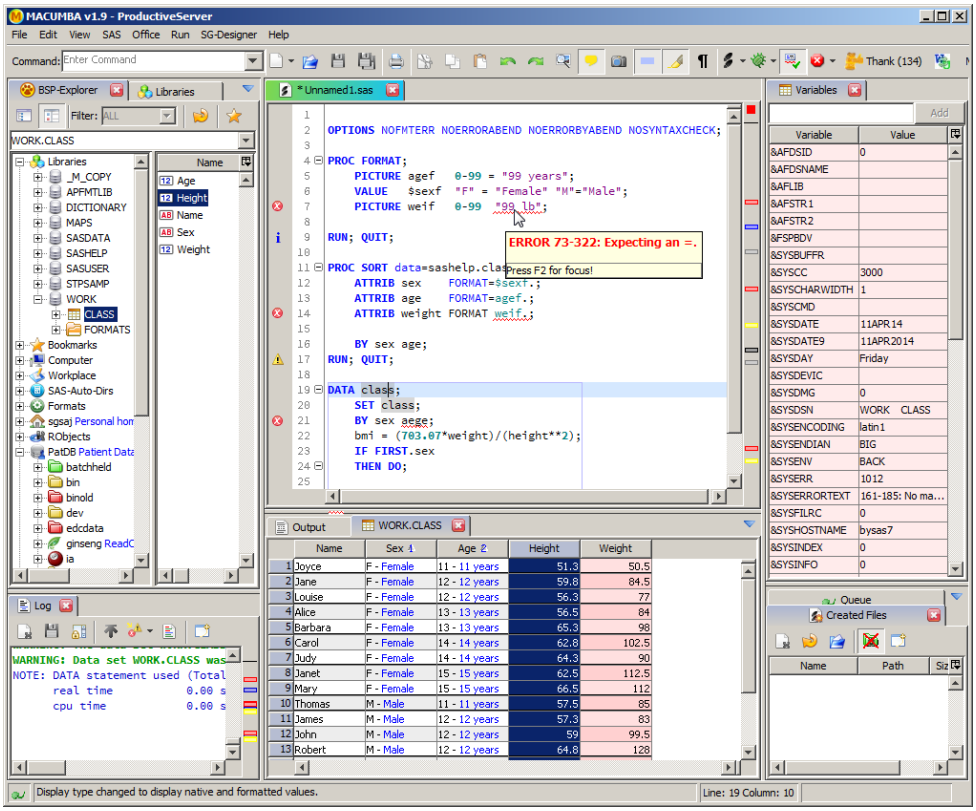

Figure 1: MACUMBA Main Screen

This paper focuses on the log alignment feature of MACUMBA that integrates all relevant log lines into the SAS program editor next to the source code line that is responsible for the log entry.

At the beginning, the general SAS log is introduced and it is explained how to manually follow the program flow in the log. As a next step, some special points are discussed, that have to be considered when designing an algorithm that should align the SAS log messages to the related program code.

At the end of this paper, the MACUMBA implementation is shown and a small example program is introduced, that can be downloaded to try the log alignment on custom SAS programs.

WORK WITH THE SAS LOG

SAS provides a very extensive logging facility and lots of system options to control the amount of log messages. This provides on the one hand a very complete overview of the executed SAS program. On the other hand, the amount of uninteresting (expected) log messages makes it almost impossible to manually find all interesting (unexpected) ones. In most situations, the majority of the SAS log messages are not of interest, because they are expected. For example the following log lines after executing a DATA step:

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

All those lines are not in the scope of this paper. Therefore only unexpected log messages are of interest. Defining a rule that matches only to unexpected log lines is not always possible. For example the following log line:

```
WARNING: No BY statement was specified for a MERGE statement.
```

This message will mostly point to a missing BY group. In most cases this is an error, but in some special cases it's correct. Another example is the following line:

```
NOTE: MERGE statement has more than one data set with repeats of BY values.
```

This might be expected in some special situations, but mostly points to a wrong by group or wrong input data. To correctly reflect these lines that might be of interest, the wording "suspicious lines" is used in this paper.

SAS flags most of the log lines with a prefix name. Therefore all log lines starting with ERROR, WARNING or FATAL are suspicious by default. Additionally specific other log messages are defined as "suspicious", too. Those are for example:

```
NOTE: MERGE statement has more than one data set with repeats of BY values
NOTE: Numeric values have been converted to character values at the places
given by <pos>
NOTE: Invalid <argument_number> argument to function <function> at <pos>
INFO: The variable <var> on data set <dat1> will be overwritten by data set
<dat2>
```

All these log lines are "just" note or info lines, but they might point to the cause of some later errors or (even worse) wrong outcome. Which log lines are to be considered suspicious and which not, has to be defined manually by the company, or by the people, that are doing the validation.

SAS LOGGING OPTIONS

SAS provides a lot of system options that interfere with the SAS log facility[1]. The most common options are the following:

- NOTES – define if (or if not) SAS notes should be output to the SAS log.
- SOURCE(2) – define if (or if not) SAS source code lines should be output to the SAS log.

If for example no note or source lines are output to the SAS log, it is almost impossible to find the cause of an error message. But even if NOTES, SOURCE and SOURCE2 are active, not all suspicious log lines will be shown. At least the options MSGLEVEL=I and MERGENOBY=WARN should also be set to receive additional messages.

A lot of other SAS system options can also influence the SAS log. For example the debugging options like FULLSTIMER, MLOGIC, SYMBOLGEN and MPRINT, but those will not be explained in more detail here.

And even if all these options are active, it is still possible to use the PRINTTO procedure to route parts of the log to a different destination. Here the SAS system option ALTLOG is of special interest for the automated SAS log processing. Specified at startup, this option allows pointing to a file that receives the full SAS log. This is especially useful in a GxP environment, to ensure all ERROR and WARNING log lines are found during validation.

[1] SAS Output: The SAS Log (SAS Institute Inc., 2014)

IDENTIFICATION OF SUSPICIOUS LOG LINES

Identifying the FATAL, ERROR and WARNING lines is relatively easy and can be done by a simple text search. The identification of the other suspicious log messages is more complex because of the high amount of different log lines. Therefore a special tool (like a SAS macro) should be used to parse the SAS log and filter all related lines. This approach was already discussed in lots of papers and multiple SAS macros (like %check_log[2] or %CheckLog[3]) are already available to serve this need.

From developer perspective, this identification of the suspicious log lines is just half of the way. The other half is the determination of the source code line that is responsible for the suspicious line, to either fix it (in case of wrong code) or to ensure, that the log line is expected at this position.

MERGING OF BROKEN LINES

From time to time, SAS proves its age and the strong alignment to the mainframe times. When handling the log, this can be seen with long log messages. SAS still takes the LS and PS options into account for the log file. While the PS option can almost be disabled by using OPTIONS PS=MAX (32767 lines per page) the LS option only allows a maximum line size of 255 characters per line. If the listing destination should be used to generate printable output, the log is simply formatted in the same way that is required for the listing output.

When scanning the log for suspicious lines, this is an important point that has to be considered. Some of the suspicious lines are longer than allowed by the LS option, for others the length depends on a displayed file name, variable name or similar. In case a log line contains more characters than the allowed line size, SAS breaks it into multiple lines. For example the log line

```
NOTE: Numeric values have been converted to character values at the places given by: (Line):(Column).
```

might be split into

```
NOTE: Numeric values have been converted to character
      values at the places given by: (Line):(Column).
```

and would therefore not be found when searching for the complete text.

SAS uses the rule, that a line is split after the last space that is not behind the allowed line size. Additionally the next line is initialized with a number of spaces to align the rest of the log message with the beginning of the previous message (without the prefix).

A tool that scans the log for suspicious lines should therefore reverse this splitting and merge long lines back to one. Unfortunately SAS only indents internal messages. Custom messages (e.g. by using PUT or %PUT) are split correctly after the last space that can be placed into the line, but the following line is not indented using spaces.

Source code lines are split a little differently and sometimes are repeated if multiple issues are found in the same line, but this is not handled in more details here.

ALIGN THE LOG

In modern times a full study evaluation can contain hundreds of SAS programs with thousands of lines of code and not all issues are found in development when running just short parts.

When running all study programs together for the final evaluation, the resulting log can grow to multiple megabytes and hundreds of thousands of lines.

When errors occur in this final run a manual search for the code position can be cumbersome. Especially when executing big programs with lots of includes or lots of similar code parts. In some cases the program flow needs to be followed in the log from the beginning, or at least for a longer period, to find the SAS code line, related to a log message.

Here it would be a good help, to have a tool that does the alignment between the SAS log and the SAS program and that provides an easy overview of the suspicious log lines aligned to the SAS program.

---

[2] Check_log macro: http://listserv.uga.edu/cgi-bin/wa?A2=ind0206b&L=sas-l&P=29795
[3] CheckLog macro: http://sas.cswenson.com/checklog

MANUAL LOG ALLIGNMENT

For a human it's relatively easy to follow the SAS log file and keep track of the related SAS program line. Even when source code lines are missing or source lines of included programs interfere. A human uses implicit strategies by understanding the available log lines to follow over those regions. For example a human can easily oversee the code in Table 1 and know that the WARNING is related to the third DATA step. An automatic tool might either align this warning to the one available source line at the beginning, or at least to the second DATA step.

SAS program	SAS log

`OPTIONS NOSOURCE;`
`%MACRO sendLog(path);`
`  PROC PRINTTO LOG=&path.;`
`  RUN;`
`%MEND;`
 	
`DATA class;`
`  SET sashelp.class;`
`RUN;`
 	
`%sendLog("/dev/null")`
 	
`DATA class;`
`  SET class(FIRSTOBS=15);`
`RUN;`
 	
`%sendLog(log)`
 	
`DATA class;`
`  SET class(FIRSTOBS=15);`
`RUN;`
	`1  OPTIONS NOSOURCE;`
	
	`NOTE: There were 19 observations read from the data set SASHELP.CLASS.`
	`NOTE: The data set WORK.CLASS has 19 observations and 5 variables.`
	`NOTE: DATA statement used (Total process time):`
	`  real time  0.00 seconds`
	`  cpu time  0.01 seconds`
	
	`NOTE: PROCEDURE PRINTTO used (Total process time):`
	`  real time  0.00 seconds`
	`  cpu time  0.00 seconds`
	
	`WARNING: FIRSTOBS option > number of observations in WORK.CLASS.`
	
	`NOTE: There were 0 observations read from the data set WORK.CLASS.`
	`NOTE: The data set WORK.CLASS has 0 observations and 5 variables.`
	`NOTE: DATA statement used (Total process time):`
	`  real time  0.00 seconds`
	`  cpu time  0.01 seconds`

Table 1: Example SAS code and related log

Unfortunately humans are slow, so this manual strategy might take a lot of time for big log files.

AUTOMATIC LOG ALIGNMENT

A computer can do the alignment very fast, but it's almost impossible to implement a full understanding of the log file. This results in specific requirements that have to be fulfilled by the SAS log, to have the computer being able to align the log to the related program.

Those requirements are:

- Always keep system options NOTES, SOURCE and SOURCE2 enabled (at least outside of macro executions)
- Do not manually create log lines that look like SAS source code lines or include notes (e.g. through PUT or %PUT commands.
  This is even true for the NUMBER option, that should be disabled (NONUMBER) because the headlines can be mixed up with source lines.
- Either do use the ALTLOG destination or don't route parts of the SAS log to other files (e.g. by using PROC PRINTTO LOG="...")

An additional challenge in the automated alignment is related to the placement of log messages related to the source code line placement in the SAS log.

In most cases the SAS code line will be found before the related suspicious log lines, but in some special cases (especially when using SAS macro code) the suspicious line is output before the related source line, or not even nearby. Especially with mismatching %DO and %END commands, it often occurs that the respective ERROR messages are found somewhere in the log.

If all requirements are fulfilled, then an easy algorithm can be applied. Scanning the log line by line and perform the following:

- If a source line is found, move one line down in the program code and set this line "active".

- If a suspicious line is found, align it to the currently “active” line in the program.

This algorithm even does a pretty good alignment, but as soon as one of the requirements is not kept, it provides wrong results.

An easy improvement can be to check the content of the source code lines in the log against the content in the SAS program. Additional improvements can be received by checking the line numbers and the line type:

- Check if the source line in the program starts with the code, the source line in the log contains
- SOURCE2 lines can only be issued from an included program (or a CALL EXECUTE statement)
- SOURCE lines can only be issued from a directly executed program.
- If the line number of a SOURCE2 line starts again from 1, this indicates mostly a CALL EXECUTE generated line that can be ignored.
- If the line number is bigger than the last line number+1, check also source lines further down in the program for a match

An algorithm that follows these rules can produce usable results even for SAS log files that do not contain the full code or not all source lines.

SPECIAL ALIGNMENTS

The general log alignment can be done based on the algorithm, that a suspicious line belongs to the previous source code line.

In some cases, this assumption is wrong and the log line belongs to a previous source code line but was issued after another source code line was executed.

For example the following log line is related to specific code lines inside of a DATA step but is found after the RUN statement of the DATA step:

```
NOTE: Invalid third argument to function SUBSTR at line 4 column 11.
```

Or even the following ERROR message can be found after a RUN command was executed, but would correctly belong to a BY command:

```
ERROR: BY variable aege is not on input data set WORK.CLASS.
```

In this case a content related alignment strategy has to be implemented to receive better results. Therefore these messages can be separated into two groups:

1. Messages containing an explicit source code position, like:
   - ```
     NOTE: Numeric values have been converted to character values at the
     places given by: (Line):(Column).
           15:7
     ```
   - ```
     NOTE: Division by zero detected at line 6 column 8.
     ```
2. Messages that can be aligned based on the content, like:
   - ```
     ERROR: BY variable aege is not on input data set WORK.CLASS.
     ```
   - ```
     NOTE: The DROP and KEEP statements are not supported in procedure steps
     in this release of the SAS System.
     ```

Additionally some messages are a mixture of both. For example the log message

```
NOTE: Invalid third argument to function SUBSTR at line 4 column 11.
```

contains the explicit source code position (4/11) and an additional info (third argument).

The automatic alignments of the first type can simply be done by parsing the log line for the related position in the source code and aligning the message respectively.

The alignment of messages from the second group is more complicated. Here no general parsing can be done. Special strategies have to be implemented for each suspicious line. For example the message

```
ERROR: BY variable aege is not on input data set WORK.CLASS.
```

should be aligned to a `BY` command that names the given variable (aege) and is located after a SET or MERGE command that names the given data set (WORK.CLASS). Here every procedure has its own suspicious lines to be considered, so that a complete implementation is almost impossible and much too expensive.

A good way is to implement the most common messages for the most often used procedures. Especially in short procedures, like the SORT procedure, the benefit of special alignments is not that high as it might be in a big DATA step.

Another special point, when implementing an automated log alignment, is the SAS underlining functionality. For most syntax errors, SAS provides a more detailed hint, where the error is located in the code, by underlining the respective location like in the following example:

```
1          data _NULL_;
2            a=1 2; b=2 3;
                 ¯¯      ¯¯
                 22      22
ERROR 22-322: Syntax error, expecting one of the following: !, !!, &, *, **, +,
-, /, <, <=, <>, =, >, ><, >=, AND, EQ, GE, GT, LE, LT, MAX, MIN, NE, NG, NL,
OR, ^=, |, ||, ~=.

3          run;
```

For this type of messages, it seems to be clear where to align it, but for big procedures or DATA steps, the error message can occur way below the underlining mask and the previous source line might be a broken line or partially repeated, so a one on one mapping is not always possible. Here the best strategy is to align the source line in the log with the real code in the program and add first add the underline with the error number. As soon as later on the explaining log line is found, the message can be aligned.

MACUMBA IMPLEMENTATION

Beside the general “Log View” and a special “Suspicious Log View”, that only shows log lines considered suspicious, the MACUMBA IDE implements all of the introduced strategies to provide an automated log alignment into the SAS programming window.

Like previously explained, the suspicious log lines are aligned to the related source code line. A special alignment for late issued messages is implemented to correctly align log lines that are issued further down, e.g. after the RUN command, to the real code line. And, if possible, the alignment is even done to the correct characters.

The source lines, that produced suspicious log lines, are marked by a corresponding symbol on the left hand side row header and in the right hand side indicator bar, like seen in Figure 2. All markings provide the corresponding log message as a tool tip to provide fast access to the issue.

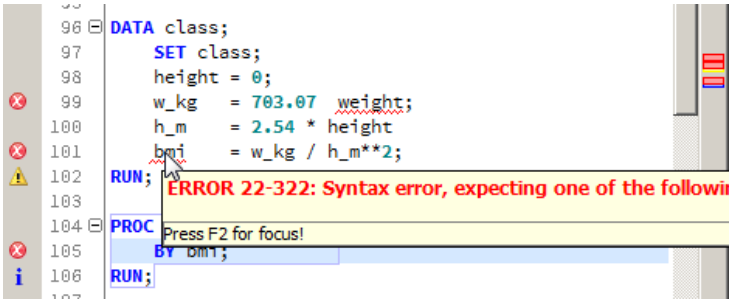

Figure 2: Log alignment in MACUMBA

Additionally a linking between the code and the log is implemented, so that a click on the row header will directly scroll the Log View to the corresponding line.

As a special feature while developing SAS code, MACUMBA contains a special code execution mode. In this mode every SAS code line is executed independently and the related SAS log is retrieved in between. This allows an exact alignment between the suspicious log lines and the related source code.

YOUR TAKE HOME

As mentioned in the introduction, the MACUMBA IDE is an in-house development at Bayer and it is not available to the public. Nevertheless, the introduced feature is very helpful while doing development, validation or just log review of a productive execution.

As a special gift, for this paper, a tool is created that implements most of the introduced alignment strategies.

The “Simple SAS GUI” (see Figure 3) allows opening a SAS program together with the SAS log and aligns the log to the program code. A SAS code execution feature (batch and pseudo interactive) is also implemented to allow some kind of playing around with the SAS code.

The tool is implemented in pure Java[4] and supports local SAS execution in Windows, Linux and UNIX as well as remote execution through an external tool like plink[5].

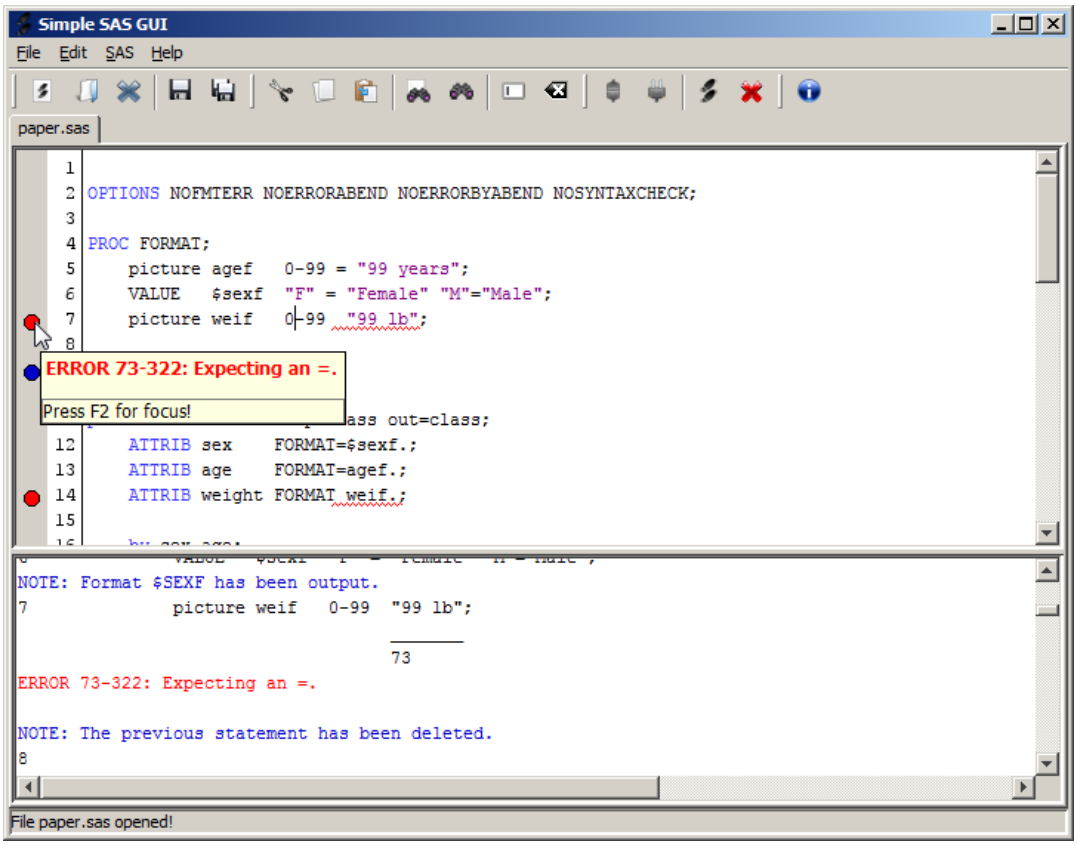

Figure 3: Simple SAS GUI Screenshot

DISCLAIMER (SIMPLE SAS GUI)

The “Simple SAS GUI” is provided under the terms of the Apache License V2.0[6]

---

[4] Java: http://www.java.com
[5] PLINK (PUTTY): http://www.chiark.greenend.org.uk/~sgtatham/putty/
[6] Apache License V2.0: http://www.apache.org/licenses/LICENSE-2.0

See the License for the specific language governing permissions and limitations under the License.

DOWNLOAD

If you agree to the Apache License, Version 2.0 as described above, you can download the “Simple SAS GUI” at http://www.magicshadow.de/pharmaSUG_2014/

CONCLUSION

The SAS log is a great help to follow over a program execution and ensure the program execution went in the same way, as expected by the developer. Nevertheless the amount of log lines makes it hard for a human to manually oversee the complete log, find all unexpected entries and align them to the correct position in the executed SAS program.

Therefore extra tools should be used to at least check the SAS log for suspicious entries. The automated alignment of the suspicious lines into the executed SAS code makes it afterwards really easy to find the related code point and fix it.

REFERENCES

Apache Foundation. (2004, 01). *Apache License Version 2.0*. Retrieved 04 10, 2014, from http://www.apache.org/licenses/LICENSE-2.0

DeShon, J. (2002, 06 12). *SAS-L archive (Post 29795)*. Retrieved 04 10, 2014, from http://listserv.uga.edu/cgi-bin/wa?A2=ind0206b&L=sas-l&P=29795

Oracle. (2014). *java.com*. Retrieved 04 10, 2014, from http://www.java.com

SAS Institute Inc. (2014). *SAS Output: The SAS Log*. Retrieved 04 10, 2014, from https://support.sas.com/documentation/cdl/en/lrcon/62955/HTML/default/viewer.htm#a000998454.htm

Swenson, C. (2012, 05 22). *About CheckLog - Chris's SAS Macros*. Retrieved 04 10, 2014, from http://sas.cswenson.com/checklog

Tatham, S. (2014, 03 14). *PuTTY: a free telnet/ssh client*. Retrieved 04 10, 2014, from http://www.chiark.greenend.org.uk/~sgtatham/putty/

ACKNOWLEDGMENTS

All this would not have been possible without the support of Elena Glathe. Your challenging requests push me every day a little further.

CONTACT INFORMATION

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

Name:	Michael Weiss
Enterprise:	Bayer Pharma AG
Address:	Muellerstr. 178
City, State ZIP:	Berlin, 13353
E-mail:	Michael.weiss@bayer.com