Paper 098-2012

An Advanced, Multi-Featured Macro Program for Reviewing Logs

Chris Swenson, UW Health, Madison, WI, USA

ABSTRACT

An important, time-consuming task for any level of SAS® user is reviewing the log for issues. Further, if there is no notification or plan in response to the notification, essentially no review occurred. There are several macros that check the SAS log for issues, and here we present a new macro program that adds several new features to the existing paradigm that assist with responding to issues. The macro includes capabilities to check internal and external SAS and logs not from SAS; to include keywords and/or exclude phrases from the search; to modify the notification behavior using a pop-up message, email, and/or sound; to “recreate” the log in the current interactive SAS session for review; and to abort the program should issues arise.

INTRODUCTION

SAS users are often beset with a never-ending problem: Accompanying any powerful program is usually a very long log, and identifying issues within the log can be time-consuming, tedious, and prone to error. Further, new SAS users often do not know what to look for. (Not all potential issues are in red or green text!)

As an analyst for a health insurance company, I was asked to develop a complex program that would allow a non-SAS programmer to run the program on demand. It was apparent that this program had to not only be robust to any normal issues, but also be able to notify the user if anything went awry. As a result, I developed the first version of the CheckLog macro program.

CheckLog is able to review the current log, an external SAS or non-SAS log, or a directory of logs for issues, and it issues messages to the user regarding the general status of what it has reviewed. There are many options for its behavior, including the capability to include keywords and/or exclude phrases from the search; to modify the notification behavior using a pop-up message, email, and/or sound; to “recreate” the log in the current interactive SAS session for review; and to abort the program should issues arise.

CheckLog does not entirely solve the problem. The user must take the information from CheckLog and develop a plan for reacting to the information supplied, whether it is dynamic code, manual intervention, or some other solution. Further, CheckLog cannot identify all issues within a given program (e.g., logical errors). Nonetheless, CheckLog automates a particularly tedious task and allows the SAS user to spend more time on more complex issues.

USAGE

The macro is available at the following website: http://sas.cswenson.com/checklog. I recommend setting up the macro as an autocall macro, which would require the user to modify the SASAUTOS= option and save the CheckLog source code in a particular directory. For more documentation on autocall macros, see SAS Institute Inc. (2012). For demonstration purposes, the reader may execute the following code to immediately include CheckLog in their current SAS session:

```
/* Download and Compile CheckLog */
filename code url "http://sas.cswenson.com/downloads/macros/CheckLog.sas";
%include code / nosource;
filename code clear;
```

The simplest method of execution is to check the current log by writing the following code:

```
%checklog;
```

The user can also set up the macro to be run from a keyboard shortcut. To set up a keyboard shortcut in Base SAS, press F9 and enter the following text in the Definition:

```
gsubmit "%checklog;"
```

I recommend using SHIFT+F6, since F6 is the default shortcut for the log window.

Since CheckLog identifies particular issues by simple keywords (e.g., “Error”), it is best to avoid using these terms within code, especially in multi-line comments. CheckLog attempts to exclude keywords within comments, but does not attempt to identify multi-line comments. If usage of a keyword is necessary, mask the term using the %STR function. The following code shows an example of masking a keyword in comments and code:

```
/* The following DATA step checks for records in the EMPTY data set, and
   generates an (e)rror message when records are found. This comment is a
   multi-line comment with a keyword masked using parentheses. */
data _null_;
    set empty;
    if _n_=1 then put "%str(E)RROR: There are records in the data set EMPTY.";
run;
```

Note that for SAS to actually use the %STR function in the text, double quotes are necessary to resolve the macro function.

In unavoidable situations (e.g., tables with the keywords in the name), the user can use the EXCLUDE option (see below) to reference a data set that contains a list of messages to ignore.

In addition to typing up the code to use CheckLog, it can be launched from a Windows shortcut or context menu. The scripts and instructions for set up are available online at http://sas.cswenson.com/downloads/scripts. The shortcut generates a pop-up menu in which the user can paste in the directory and filename of the log to review. The context menu script allows the user to run CheckLog using the Send To menu. Both require a Windows operating system and Visual Basic. In the future, I plan to rewrite the context menu script to run on any operating system.

Users can also subscribe to updates to the macro via email. To subscribe, visit the CheckLog website, locate the subscription link, and enter an email address. An invitation will be sent to the email address for the user to verify the subscription.

DOCUMENTATION

The CheckLog macro program is set up with a great deal of metadata in the header of the program, including the list of arguments and their descriptions, the keywords and issues identified, and a revision history. Following the header, each major section of the macro program is identified with its own header that includes the word “Section” followed by the section number (and title), allowing the user to easily find a section by searching for “Section X” using the desired section number. A list of sections is available at the top of the main header.

KEYWORDS AND MESSAGES

The following keywords and phrases are identified as issues by CheckLog. Some of the phrases could have been truncated; however, since users can write their own messages to the log, I thought it best to avoid phrases that were too short in order to avoid user-written notes and therefore false positives.

- Keywords: “Error”, “Warning”, “Invalid”, and “Uninitialized”
- Conversion:
  - “Numeric values have been converted to character values…”
  - “Character values have been converted to numeric values…”
- Formatting: “At least one W.d format…”
- Input
  - “Lost card…”
  - “New line when input…”
  - “One or more lines were truncated…”
- Math
  - “Division by zero detected…”
  - “Mathematical operations could not be performed…”
  - “Missing values were generated…”
- Merging: “Merge statement has more than one data set with repeats of by values…”
- SQL
  - “The execution of this query involves performing one or more Cartesian product joins that cannot be optimized…”
  - “The query requires remerging summary statistics back with the original data…”
- Syntax

- "A case expression has no else clause…"
- "The meaning of an identifier after a quoted string may change…"

In my experience, the "format" issue often appears at random and is not always indicative of a data issue. It appears this happens when using the WORK library, since SAS provides fewer resources to temporary libraries than permanent libraries. However, when the issue is not random, it does usually indicate a problem with the length of the format.

Occasionally, the SQL messages are intended by the user. However, this style of coding can mask underlying issues with the data, and I believe it is best to avoid this style. Additionally, administrators can actually turn remerging into an error instead of allowing it to execute. For more on the topic of SQL remerging, see my blog post at the follosing URL:

http://blog.cswenson.com/2012/01/sql-remerging.html

EXCLUDED MESSAGES

The following phrases are excluded from identification as issues. All keywords found in comments are excluded. However, CheckLog does not attempt to reconcile multi-line comments, which brings me back to a previous point: avoid use of the keywords or mask their use. Masking the keywords in multi-line comments avoids identification of the comment as an issue (see above for an example). The specific messages excluded include variable messages, a couple of SAS-generated messages, and library messages which will result in more serious issues if they are used.

- Variable messages
  - "_ERROR_ = 0"
  - "IF _ERROR_"
  - "Set the error detection macro…"
- SAS messages
  - "The macro … completed compilation without errors."
  - "Your system is scheduled to expire on..."
  - "Errors printed on page…"
- Library messages
  - "Error in the libname statement…"
  - "Only available to users with restricted session privilege…"
  - "The account is locked…"
  - "Unable to clear or reassign the library…"
  - "Unable to copy the SASUSER registry to work registry…"
  - "User does not have appropriate authorization level for…"

The last set of library messages was excluded because in the environment in which I developed the macro, a default autoexec ran with SAS that assigned several libraries I did not have rights to access. Since I could not modify the autoexec or disable the attempt to assign these libraries, these messages were useless when identified as issues. Further, libraries may or may not be used in any given program, and if they are used, other errors will indicate the source of the problem.

ARGUMENTS

Given its capabilities, CheckLog has a number of arguments. The arguments generally fall into three categories: input, reporting, and operational arguments. All arguments are optional. If no options are specified, the macro checks the current log, outputs issues to Log_issues, and issues a pop-up message with the status of the log.

For users new to macro programs, positional arguments do not require the use of the argument name when specifying a value, but they have to be in the same order specified by the macro program. Keyword arguments require the use of the argument name followed by an equal sign and the desired value (e.g., pm=Y). In my macros, any arguments that are "binary" ("yes" and "no"), can be specified with the full length value or 1 character ("yes" or "y" or even "yup" work). Here are three examples of the use of positional and keyword arguments using CheckLog:

```
/* Positional argument */
%CheckLog(C:\dir\example.log);

/* Keyword arguments */
%CheckLog(pm=E, abort=Y);

/* Both types of arguments */
%CheckLog(C:\dir\example.log, abort=Y, pm=E);
```

INPUT ARGUMENTS

The following arguments manage what is sent to CheckLog to include or exclude from the review.

- LOG – The log to check, either blank (current log), a specific file (with the directory), or a directory that contains logs. Note that the extension "log" is searched for in a directory, and can be overwritten by the EXT= argument (see below).
- EXT= – When reviewing directories, this argument specifies the extension to use to identify files to include in the review, defaulted to LOG. Another common alternative is TXT.
- KEYWORD= – This argument can be used to list additional keywords to include in the search for issues. Keywords used in non-SAS logs or in user-specified issues can be added. For example, the terms CAUTION or ALERT may be used in non-SAS logs or user-specified issues. Note: The order of the words listed matters. The words listed first will be searched for last and will overwrite any other words found prior. Thus, the first word should have the highest priority. Of course, the keywords in the macro program will overwrite user-specified keywords (e.g., "ERROR" will always trump the words in the KEYWORD= argument).
- EXCLUDE= – As noted above, this argument can be used to exclude specific phrases as issues. The user specifies a data set that contains a variable named LogText, which contains specific messages to exclude. If the variable name is not LogText, specify the variable named after the data set name in the argument, followed by a space (e.g., work.exclusions list).

Initially, I used the EXCLUDE= option to exclude SAS software expiration warnings, but I eventually wrote that exclusion into CheckLog. The variable named selected by default, LogText, is the same variable name used in the output data set from CheckLog. Therefore, once a user identifies a false issue, they can use the output from CheckLog to write those messages out to a permanent data set to use as an exclusion data set.

The EXCLUDE= argument can be used when non-issues are encountered that cannot be filtered based on a simple rule (e.g., a data set name unavoidably contains the word "error"). It is NOT recommended that this feature be used to circumvent code revision for easily-avoided issues. Note that data set options can be applied with this argument, so a master file with exclusions can be used and filtered:

```
%CheckLog( C:\dir\test.log, Share.Log_exclusions(where=(Program="Claims")) Issue );
```

REPORTING ARGUMENTS

The following are reporting arguments, in which the macro notifies the user (e.g., via a data set or a pop-up message) whether any issues were found. Some of these arguments require simple Yes/No values, but some have a third option, E, which indicates that the behavior should occur only when issues (Errors) are found.

- OUT= – The name of the output issues data set, defaulted to Log_issues. This argument is automatically set to Logs_w_issues when checking a directory of logs.
- PM= – A pop-up message is used when running SAS in interactive mode to notify the user of the status of the log. This argument determines the behavior of this pop-up. Valid values: N = No, never display a pop-up message; Y = Yes, always display a pop-up message (default); E = Error-only, display a pop-up only when issues are found.
- SOUND= – Optionally, a sound can be output, which varies depending on the status of the log. A sound like "Hurray!" and "Uh-oh" were selected based on the status of the log. This option can only be (successfully) used with Microsoft Windows©. Valid values: N = No, never display a pop-up message; Y = Yes, always display a pop-up message (default); E = Error-only, output a sound only when issues are found.
- RELOG= – This option recreates ("relogs") the SAS log as if the current interactive SAS session had run the program. It overwrites the current log with the imported log, after CheckLog outputs notes regarding the status of the log. This is best used for reviewing logs output from batch mode. Valid values: N = No, do not overwrite the current log (default); Y = Yes, overwrite the current log with the imported log; E = Error-only, overwrite the log only when issues are found.
- TO= – An email can be sent listing the types of issues encountered. Specify an email address in this field to output an email. Note that email addresses are not validated by CheckLog, and the user is responsible for

gaining authorization to use email and permission to send email to the recipient. Email support must be set up in SAS to use this function.

- CC= – Specifies a CC email address. See above. Note that BCC is not supported for reasons regarding privacy noted below. The CC argument will not work without the TO argument.

A Warning Regarding Email and Privacy Laws

A warning regarding email and privacy laws, such as the Health Insurance Portability and Accountability Act of 1996 (HIPAA): Given the sensitivity of healthcare data and the transparency and ease-of-interception of plain-text emails, the CheckLog macro program, by design, does NOT included *specific* issue messages or related notes, nor does it include BCC recipients. Log lines with the term "error" can include patient information, and emailing this information would likely violate privacy laws, such as HIPAA. Any users who attempt to overcome this limitation risk personal responsibility for violating laws such as HIPAA. For more information on HIPAA, please visit the U.S. Department of Health and Human Services website at http://www.hhs.gov/ocr/privacy/.

OPERATIONAL ARGUMENTS

The following arguments modify how CheckLog operates. Using these options, the user can fine-tune how CheckLog operates and output additional information.

- DIREXT= – When reviewing directories, this argument specifies whether to include the full directory and extension of the file in the output data set. By default, only the name of the log (e.g., "test" for the file "C:\test.log") is included in the output data set. When DIREXT is set to Y, it adds both the directory and extension. Valid values: N = No, do not add directory and extension (default); Y = Yes, add directory and extension.
- SHADOW= – Occasionally the user may wish to check a log for a program that is still running. In this situation, it is necessary to make a copy of the log before associating with it using the FILENAME function; thus a copy of the log is necessary before importing it. This argument is now set to Y by default, and to maintain backward-compatibility, it is left in the list of arguments. Users can still turn it off if desired. Valid values: N = No, do not make a copy of the log; Y = Yes, make a copy of the log (default).
- ABORT= – Determines whether or not the following program is aborted. This is useful to halt a long program and alert the user that there are issues. Valid values: N = No, do not abort program when issues occur (default); Y = Yes, abort program when issues occur.
- TEST= – To test CheckLog, set the test argument to "test" (i.e., test=test). During normal usage, CheckLog turns off the log in order to avoid detecting issues and phrases within its own execution and to avoid outputting irrelevant processing in the log. In test mode, it leaves the log on and does not delete temporary data sets. Additionally, the extra data set Log_keywords is output for troubleshooting.

CAUTION: The ABORT= option may have undesirable results, especially in SAS 9.1. Please use it with care and thorough testing. In version 9.1.3, it has been identified that when the abort option is specified and used in a program, then the CheckLog macro program is used from a keyboard shortcut, the SAS software crashes. There is no known workaround and it appears to be a bug in SAS 9.1.3. To abort macro programs, it is much more stable to use the ISSUES macro variable that is output from CheckLog, as in the following statement, which ends the macro program but not the code that follows the macro call:

```
%if &ISSUES>0 %then %return;
```

OUTPUT

Upon completion, CheckLog can generate a few data sets, notifications, and a global macro variable containing the number of issues found, called ISSUES. The following data sets can be generated:

- Log – The imported log.
- Log_issues – The statements in the original log associated with issues. The name is set by the OUT= option. The number of observations in Log_issues is equal to the number stored in the ISSUES macro variable.
- Log_keywords – In test mode, this data set includes all statements with keywords and phrases identified during the check, including excluded phrases. This data set is best for troubleshooting the macro itself.
- Log_summary – For emails, this data set is generated to send a summary of the issues in the email body.
- Logs_w_issues – For directories, this data set stores the list of logs in a directory associated with issues.

By default, a pop-up message appears in interactive SAS, with a message regarding the status of what was checked, along with two data sets (when checking a single log), Log and Log_issues (or Logs_w_issues in directory mode). With additional arguments specified, an email can be sent, a sound played, the log recreated, or the program aborted.

CheckLog provides a number of ways to respond to issues. Simply notifying the user that something is amiss, by pop-up, sound, or email, is the simplest step. The RELOG= option helps the user read the log using the enhanced editor colors. A user could write a program with CheckLog that detects issues using the ISSUES macro variable or the Log_issues data set, and the program responds by aborting, re-running a particular section of code, branching the code into different strategies for handling the issue, halting the code for the user to return, or any number of interventions. A plan is essential for a successful review; otherwise, there was no purpose in conducting the review.

CONCLUSION

As robust as CheckLog is, it will never be able to identify all problems within a program. There are several limitations:

- Users need to plan to respond to issues. To do otherwise essentially ignores the output and brings in to question the utility of completing the check.
- Not all issues can be found, either because they are not known or they cannot be checked programmatically. For example, the validity of IF/THEN statements or the removal of valid duplicates using PROC SORT with the NODUPKEY option must be manually validated.
- Some users may disagree with the list of issues I have compiled. These users are free to comment out those issues from the macro.
- False positive results can be found in multi-line comments. Occasionally the format length issue is a false positive result as well.
- The names of files in a directory should be valid SAS names; otherwise, it is possible that two separate logs conflict on name. CheckLog attempts to remove invalid characters to use as the data set name, but it's possible that confusing or conflicting results could be generated.
- CheckLog cannot always execute as expected. In batch mode, it cannot be the first step in the program. Otherwise, it will not find any records in the log. It appears that either a DATA step or PROC need to be run before CheckLog is called. Obviously, CheckLog will not execute within a piece of code if the SAS software itself crashes.

New versions of CheckLog are released when new issues are identified, which mitigates the limitation regarding unknown issues. Further, users can subscribe to updates by news feed or email. Any interested readers can subscribe on the CheckLog website. This process is also used to fix bugs or add features.

The limitation that arises when the SAS software crashes can be overcome by splitting the task into two processes: one that runs the code and another that checks the log of the other code. CheckLog can still be used within the program at various stages, but the final review is completed by an independent process that will only fail if the operating system fails.

A robust way to set this up is to output all logs to the same directory and schedule SAS to run CheckLog on new logs on a regular basis (e.g., weekdays). An input data set could be used to drive its execution and set different people to notify by email about the status of different logs. The SAS administrator, the author, and even customers of the data/reports can be notified of the status of the last execution of the program.

Programmatic checks such as CheckLog can increase efficiency in the development and production stages, however, in between these stages, good peer review is still necessary to review the code for issues that cannot be programmatically check and to ensure the process meets standards and specifications, good coding practices, proper report formatting, and many other aspects that can only evaluated by robust peer review.

In the end, there are three key takeaways from this discussion:

1. It is important to review the log.
2. A programmatic review can increase efficiency and code dynamism, but peer review is still necessary.
3. Planning a response to issues is key to completing the code review.

CheckLog is a useful tool for addressing issues, and users equipped with CheckLog are ahead in development efficiency and the capability to write responsive, dynamic code.

REFERENCES

SAS Institute Inc. (2012). SAS 9.2 Macro Language: Reference: Saving Macros in an Autocall Library [URL]. http://support.sas.com/documentation/cdl/en/mcrolref/61885/HTML/default/viewer.htm#a001328769.htm

ACKNOWLEDGMENTS

Special thanks to the reviewers of the CheckLog macro program, especially Iryna Feldman, the initial peer reviewer, and Hung Tam Nguyen, who helped test the macro on Unix. Additional thanks to Lauren Lake, my SAS Global Forum mentor.

RECOMMENDED READING

- Augustine, Aaron. (2010). SAS® Code Validation: L.E.T.O Method [PDF]. http://support.sas.com/resources/papers/proceedings10/083-2010.pdf
- Foley, Malachy J. (2005). MERGING vs. JOINING: Comparing the DATA Step with SQL [PDF]. http://support.sas.com/resources/papers/proceedings09/036-2009.pdf
- Kuligowski, Andrew T. (2005). In Search of the LOST CARD [PDF]. http://www2.sas.com/proceedings/sugi30/058-30.pdf
- Li, Tianshu & Troxell, John K. (2001).A Macro to Report Problematic SAS Log Messages in a Production Environment [PDF]. www.nesug.org/proceedings/nesug01/cc/cc4008.pdf
- Slaughter, Susan J. & Delwiche, Lora D. (1997). Errors, Warnings, and Notes (Oh My): A Practical Guide to Debugging SAS Programs [PDF]. http://www2.sas.com/proceedings/sugi22/BEGTUTOR/PAPER68.PDF
- Smoak, Carey G. (2002). A Utility Program for Checking SAS® Log Files [PDF]. www2.sas.com/proceedings/sugi27/p096-27.pdf

CONTACT INFORMATION

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

Chris Swenson
E-mail: chris@cswenson.com
Web: http://www.cswenson.com/