Paper 1173-2017

Check Please: An Automated Approach to Log Checking

Richann Watson, Experis

ABSTRACT

In the pharmaceutical industry, we find ourselves having to re-run our programs repeatedly for each deliverable. These programs can be run individually in an interactive SAS® session, which enables us to review the logs as we execute the programs. We could run the individual programs in batch and open each individual log to review for unwanted log messages, such as ERROR, WARNING, uninitialized, have been converted to, and so on. Both of these approaches are fine if there are only a handful of programs to execute. But what do you do if you have hundreds of programs that need to be re-run? Do you want to open every single one of the programs and search for unwanted messages? This manual approach could take hours and is prone to accidental oversight. This paper discusses a macro that searches a specified directory and checks either all the logs in the directory, only logs with a specific naming convention, or only the files listed. The macro then produces a report that lists all the files checked and indicates whether issues were found.

INTRODUCTION

Checking the logs for unwanted messages is one of the first steps in ensuring that a program was executed successfully. When running the program interactively this can be easily done by manually reviewing the log in the interactive session. However, if you execute the program in batch mode or from a 'driver' program and the logs are saved as individual files, then you need to open the log using SAS Universal Viewer or some other type of text editor in order to review the log. Regardless of the method you use, the log still needs to be perused in order to find these pesky messages.

TYPES OF LOG MESSAGES

There are a variety of log messages that may cause problems in the program and these messages need to be investigated. Some of the more common messages include:

- ERROR
- WARNING
- UNINITIALIZED
- MORE THAN ONE DATA SET WITH REPEATS OF BY
- VALUES HAVE BEEN CONVERTED
- MISSING VALUES WERE GENERATED AS A RESULT
- INVALID DATA
- AT LEAST ONE W.D FORMAT TOO SMALL

How these log messages are to be fixed is dependent on the data and project needs. 'ERROR' messages should always be corrected since this can affect the output. Additionally, the 'MORE THAN ONE DATA SET WITH REPEATS OF BY' should be fixed so that a many-to-many merge is not done in a data step. Some other messages may not have an effect on the data and may not be as critical to fix; however, that is dependent on what the client deems as critical. There are some clients that do not wish to see any unwanted log messages and request that they all be fixed.

In addition to the common messages that you may look for in the logs, the client may have a more extensive list of messages that they may not want to appear. Other types of log messages that may cause issues are:

- ORDERING BY AN ITEM THAT DOESN'T APPEAR IN

- OUTSIDE THE AXIS RANGE
- RETURNING PREMATURELY
- QUERY DATA
- QUESTIONABLE

OTHER LOG MESSAGES TO CONSIDER

When determining the list of possible unwanted log messages to build into the macro (provided in Appendix A), you may need to consider other types of messages that may cause the macro to flag the program as containing unwanted messages. Additionally, if you are aware of specific types of log messages that will trigger an issue to be reported, then it would be wise to build into the macro the types of messages that are allowed.

For example, when a SETINIT may be close to the expiration date, the log will have several types of WARNING messages. Since these WARNING messages do not have an impact on the actual execution of the program, then logic can be incorporated to look for these types of messages.

- UNABLE TO COPY SASUSER
- BASE PRODUCT PRODUCT
- EXPIRE WITHIN
- BASE SAS SOFTWARE … EXPIRING SOON
- UPCOMING EXPIRATION
- SCHEDULED TO EXPIRE
- SETINIT TO OBTAIN MORE INFO

LOG CHECKING PROBLEM

With all the various types of messages that need to be checked during the log review, one can plainly see how quickly this process of checks and fixes can become tedious and burdensome. This manual approach to reviewing these logs can lead to several other concerns.

1. It is easy to overlook a particular message in a log while scrolling through the log.
2. When searching for a particular message, it is easy to mistype the message or get the casing incorrect and thus overlooking the unwanted message.
3. If there are a lot of logs to check, it is easy to overlook a log altogether.
4. The programmer may be short on time and may just make the assumption that the program ran clean the last time and there is an output produced so it should still be fine.

THE SOLUTION - CHECKLOGS

Rather than leave the log checking to ‘chance’ and human error an automated alternative can be implemented through use of the CHECKLOGS macro (Appendix A). The macro will parse through each of the indicated logs in a specified directory looking for the unwanted messages, and then produce a report summarizing the findings.

MACRO PARAMETERS

The macro has five input parameters, with only one being required:

- ‘loc’ – location of where the log files reside.
- ‘fnm’ – optional parameter that indicates which types of files to look at; if more than one type of file is specified then it should be separated by a space.

- 'out' – optional parameter that indicates the name of the output report file that will be produced.
- 'loc2' – optional parameter that indicates the location of where the report should be saved.
- 'delm' – optional parameter that indicates what delimiter is used in the 'fnm' parameter.

For the optional parameter 'fnm', the macro will build a where clause that will be used to exclusively extract the logs of interest. For example, if all your logs are stored in the same folder and you are only concerned with viewing the logs for the tables and figures, and all the tables and figures logs have a standard naming convention (e.g., 't_#_#_#.log' for tables and 'f_#_#_#.log' for figures where '#_#_#' represents the output number) then you can use the standard portion for each type separated by a delimiter such as '~'.

fnm = t_~l_

Note that the building of the where clause does not care if the standard portion is a prefix or a suffix as long as there is a standard and that the standard is not used for other log types. So if the table logs had a naming convention of 'xxxxx_t.log' and the figure logs had a naming convention of 'xxxxx_f.log' where 'xxxxx' represents the output name, then you can still use the standard portion for each type separated by a delimiter such as '~'.

fnm = _t.~_l.

If the 'out' parameter is not specified, then the file name for the report will default to 'all_checklogs.rtf'. Ideally, if you are executing the macro to look at only specific types of files, then the name of the report should be specified so that the report is not overwritten when executed later for another file type.

If the 'loc2' parameter is not specified, then the output location for the report will default to the location where the log files reside.

If the 'delm' parameter is not specified, then the delimiter defaults to '@'. Note that this needs to match the delimiter that is used if 'fnm' is specified and more than one file name type is indicated in the 'fnm' macro parameter. For example, above 'fnm' used a '~' as a delimiter therefore 'delm' would need to be specified. Note that it is not ideal to use a space as a delimiter since the directory path and/or the file names could have spaces.

delm =~

Table 1 illustrates some sample calls of the macro and what the expected outcome is. Note that the samples in this paper are based off of the Windows environment. However, the program will work with a Unix environment as well. The user just needs to make sure the slashes in the 'loc' parameter are pointing in the correct direction.

Sample Call	Expected Outcome
`%checklogs(loc=C:\Users\user\Desktop\SGF\ Check Logs\Logs)`	• Check **ALL** the logs in the specified directory • Use default name "all_checklogs" for the report name.
`%checklogs(loc=C:\Users\user\Desktop\SGF\ Check Logs\Logs, out=all_output_logs)`	• Check **ALL** the logs in the specified directory • Use specified name"all_output_logs" for the report name.
`%checklogs(loc=C:\Users\user\Desktop\SGF\ Check Logs\Logs, fnm=table, out=Tables_Logs)`	• Check **ONLY** the logs that contain "table" in the file name • Use specified name "Table_Logs" for the report name.
`%checklogs(loc=C:\Users\user\Desktop\SGF\ Check Logs\Logs, out=Analysis Dataset and Tables Logs, fnm = ad@table)`	• Check **ONLY** the logs that contain "ad" **OR** "table" in the file name using the default delimiter "@" • Use specified name "Analysis Dataset and Tables Logs" for the report name.
`%checklogs(loc=C:\Users\user\Desktop\SGF\ Check Logs\Logs, out= Analysis Dataset and Tables Logs delm, fnm = ad~table, delm=~)`	• Check **ONLY** the logs that contain "ad" **OR** "table" in the file name • Use the specified delimiter '~' • Use specified name "Analysis Dataset and Tables Logs delm" for the report name.
`%checklogs(loc=C:\Users\user\Desktop\SGF\ Check Logs\Logs,`	• Check **ONLY** the logs that contain "table1" **OR** "table2" in the file name

Sample Call	Expected Outcome
`loc2=C:\Users\user\Desktop\SGF\Check Logs\Log Report, out=Tables 1 and 2 Logs delm, fnm = table1!table2, delm=!)`	• Use the specified delimiter '!' • Write the report to another location • Use the specified name "Tables 1 and 2 Logs delm" for the report name.

Table 1 Sample CHECKLOG Calls and Expected Outcome

NUTS AND BOLTS OF THE MACRO

So how does this macro actually work? There are a variety of steps in the macro and each step will be explained.

Determining Working Environment

The first thing the macro does is determine in what working environment the command is being executed. The macro is able to run in either Windows or Unix environment. Depending on the environment, the macro will determine what form the pipe command should take and in which direction the slash in the path name should be pointing.

During this initial part of the macro, the following two macro variables are set and will be used in the macro:

- ppmcd – represents the pipe command
- slash

If the execution environment is Windows (i.e., &SYSSCP = WIN), then following macro variables are set as

```
%let ppcmd = %str(dir);
%let slash = \;
```

If the execution environment is Unix (i.e., &SYSSCP = LIN X64), then following macro variables are set as

```
%let ppcmd = %str(ls -l);
%let slash = /;
```

What Log Files Are Being Checked?

If the 'fnm' input parameter is specified, the macro will then parse through each token in the parameter to build a where clause. If there is more than one token in 'fnm', then the either the default delimiter or the the one specified in 'delm' parameter, will be used to parse 'fnm'. The where clause will use the SAS function INDEX to look for a specific value in the filename. If no input parameter is specified, then the macro will check every log file in the specified directory.

Using the sample calls in Table 1 the associated where clauses for each call are illustrated (see Table 2).

Sample Call	Where Clause Built
`%checklogs(loc=C:\Users\user\Desktop\SGF\Check Logs\Logs)`	No ***where*** clause. Macro will check all logs in the directory.
`%checklogs(loc=C:\Users\user\Desktop\SGF\Check Logs\Logs, out=all_output_logs)`	No ***where*** clause. Macro will check all logs in the directory.
`%checklogs(loc=C:\Users\user\Desktop\SGF\Check Logs\Logs, fnm=table, out=Tables_Logs)`	where index(flog, "table")
`%checklogs(loc=C:\Users\user\Desktop\SGF\Check Logs\Logs, out=Analysis Dataset and Tables Logs, fnm = ad@table)`	where index(flog, "ad") or index(flog, "table")
`%checklogs(loc=C:\Users\user\Desktop\SGF\Check Logs\Logs, out= Analysis Dataset and Tables Logs delm, fnm = ad~table, delm=~)`	where index(flog, "ad") or index(flog, "table")

Sample Call	Where Clause Built
`%checklogs(loc=C:\Users\user\Desktop\SGF\Check Logs\Logs, loc2=C:\Users\user\Desktop\SGF\Check Logs\Log Report, out=Tables 1 and 2 Logs delm, fnm = table1!table2, delm=!)`	where index(flog, “table1”) or index(flog, “table2”)

Table 2 Sample CHECKLOG Calls and Where Clause Built

Extracting the Logs

After the pipe command is built and the where clause is determined (if applicable) the program will bring in every file within the specified directory and create a data set. This data set will only contain the logs files that are indicated by the where clause, in the event that a where clause is built. Additionally, the name of the log, the date and time can be extracted from the various tokens in the filename. When the pipe command reads in each file all the information for that file is captured on one line (one variable) and the data can be parsed to extract each component.

Table 3 below illustrates how the filename would look if executed in a Windows environment and what the log, date and time are once they are extracted. Then numtok indicates how many tokens are in the filename. This is used to determine if there are any spaces within the filename, so that the macro will correctly assign flog.

filename	flog	ftim	fdat	numtok
09/07/2016 03:00 AM 48,511 adpr.log	adpr	7-Sep-16	3:00 AM	5
09/19/2016 08:03 AM 76,849 adpsga.log	adpsga	19-Sep-16	8:03 AM	5
08/30/2016 05:43 AM 58,634 adsl.log	adsl	30-Aug-16	5:43 AM	5
08/28/2015 05:14 AM 20,028 AE.log	AE	28-Aug-15	5:14 AM	5
09/01/2015 04:41 AM 22,392 DA.log	DA	1-Sep-15	4:41 AM	5
08/26/2015 10:59 AM 19,734 dm.log	dm	26-Aug-15	10:59 AM	5
10/13/2016 08:49 AM 23,491 graph 1_1.log	graph 1_1	13-Oct-16	8:49 AM	6
10/13/2016 08:49 AM 11,934 graph 1_3.log	graph 1_3	13-Oct-16	8:49 AM	6
09/11/2015 08:06 AM 46,331 LB.log	LB	11-Sep-15	8:06 AM	5
10/17/2016 07:27 AM 24,652 table 3_1_1.log	table 3_1_1	17-Oct-16	7:27 AM	6
10/17/2016 07:28 AM 33,002 table 3_1_2.log	table 3_1_2	17-Oct-16	7:28 AM	6
09/28/2016 10:02 AM 45,680 table1-1.log	table1-1	28-Sep-16	10:02 AM	5
10/08/2015 08:21 AM 24,857 table1_2.log	table1_2	8-Oct-15	8:21 AM	5
10/05/2016 11:08 AM 91,155 table2.1.2.log	table2.1.2	5-Oct-16	11:08 AM	5
10/08/2015 08:21 AM 36,413 table3_2_1.log	table3_2_1	8-Oct-15	8:21 AM	5

Table 3 Files Retrieved and Log Name, Date and Time Extracted

Parsing Each Log File and Creating the Log Report

Once it is determined which logs are to be checked, the macro will then go through each file and check for ‘common’ undesired log messages as well as other ‘not-so-common’ undesired log messages. At this point, the user may wish to add ‘user-defined’ log messages that should be investigated. If an undesired log message is found, it is saved to a data set so it can be included in the check log report. If there are no undesired log messages, then a ‘dummy’ record is created for that file with a generic message so that the user can be confident that the log was actually checked and not inadvertently left out of the checking process.

After each log is checked a report will be generated. If the ‘loc2’ parameter is specified, then a report will be written to the indicated location otherwise the report will be written to the location where the logs reside. If the ‘out’ parameter is specified during the macro call, the value of the parameter will be used as

the name of the report otherwise the name of the report will default to 'all_checklogs'. The report will contain the name of the log, the date the log was created, the time it was created (if applicable) and the undesired log message or generic message. In addition to this, the report will contain a blank column that will allow the user to input any additional information such as a comment as to why that specific message was allowed while they are reviewing the report.

For example, in Display 1 the logs for ADSL, DM and GRAPH1_3 had a message that contained the word 'warning'. This was part of a comment so was not an actual program warning, thus the message is allowed and so a reason is manually added to the report during the review process. Furthermore, the log for LB had two messages that contained '??' (a sponsor-defined message). In the case of the first message the flagged section was part of a comment so the message is allowed and a reason is manually added. As for the other message, a comment is added indicating that for this particular deliverable it will be allowed but must be fixed for the next deliverable.

Log Name	Log Date	Log Time	Log Message	Reason Message is Allowed
ADPR	07SEP2016	03:00 AM	No undesired messages. Log is clean.	
ADPSGA	19SEP2016	08:03 AM	NOTE: MERGE statement has more than one data set with repeats of BY values.	
ADSL	30AUG2016	05:43 AM	12735 /* otherwise put "warning exendtc= " exendtc;	This is a comment. There is no issue to fix.
AE	28AUG2015	05:14 AM	No undesired messages. Log is clean.	
DA	01SEP2015	04:41 AM	NOTE: Character values have been converted to numeric values at the places given by:	
	01SEP2015	04:41 AM	NOTE: Missing values were generated as a result of performing an operation on missing values.	
DM	26AUG2015	10:59 AM	23 SITEID at the begining of the first data step to avoid warning messages. A worning	This is a comment. There is no issue to fix.
GRAPH 1_1	13OCT2016	08:49 AM	No undesired messages. Log is clean.	
GRAPH 1_3	13OCT2016	08:49 AM	WARNING: The intervals on the axis labeled "Analysis Visit (N)" are not evenly spaced.	This is a SAS generated comment. There is no issue to fix.
LB	11SEP2015	08:06 AM	402 ! ??)*/	This is a comment. There is no issue to fix.
	11SEP2015	08:06 AM	403 RESLTVALN = input(RESLTVAL, ?? best12.);	'??'is used as part of the format statement. ***To be fixed in next deliverable.***
TABLE 3_1_1	17OCT2016	07:27 AM	No undesired messages. Log is clean.	
TABLE 3_1_2	17OCT2016	07:28 AM	No undesired messages. Log is clean.	
TABLE1-1	28SEP2016	10:02 AM	6043 no non-missing values of randexc1-randexc4, so it cause warnings and errors,	This is a comment. There is no issue to fix.
	28SEP2016	10:02 AM	NOTE: MERGE statement has more than one data set with repeats of BY values.	
	28SEP2016	10:02 AM	NOTE: MERGE statement has more than one data set with repeats of BY values.	
TABLE1_2	08OCT2015	08:21 AM	No undesired messages. Log is clean.	
TABLE2.1.2	05OCT2016	11:08 AM	No undesired messages. Log is clean.	
TABLE3_2_1	08OCT2015	08:21 AM	No undesired messages. Log is clean.	

Display 1 Sample CHECKLOGS Report

CONCLUSION

Checking the logs for a deliverable is a task we all dread doing, but it must be done. It is a necessary 'evil'. However, with the CHECKLOGS macro, this process can be simplified while simultaneously producing a report that can be saved with the deliverable to help explain any log messages that were not addressed.

REFERENCES

UNIX commands

http://www.shareittips.com/data/unix.png

CMD commands

http://ss64.com/nt/

ACKNOWLEDGMENTS

Thanks to Karl Miller and Deanna Schreiber-Gregory for reviewing the paper and providing additional insight.

CONTACT INFORMATION

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

Richann Watson
Experis
richann.watson@experis.com

APPENDIX A: CHECKLOGS MACRO

```
/* retrieve all the logs in the specified directory */
%macro checklogs(loc=,  /* location of where the log files are stored       */
                 loc2=, /* location of where report is stored (optional)    */
                 fnm=,  /* which types of files to look at (optional)       */
                        /* e.g., Tables – t_, Figures – f_, Listings – l_   */
                        /* separate types of files by delimiter indicated in*/
                        /* the delm macro parameter (e.g., t_@f_)           */
                 delm=@,/* delimiter used to separate types of files (opt'l)*/
                 out=   /* log report name (optional)                       */);

  /* need to determine the environment in which this is executed   */
  /* syntax for some commands vary from environment to environment */
  /* end macro call if environment not Windows or Linux/Unix       */
  %if &sysscp = WIN %then %do;
    %let ppcmd = %str(dir);
    %let slash = \;
  %end;
  %else %if &sysscp = LIN X64 %then %do;
    %let ppcmd = %str(ls -l);
    %let slash = /;
  %end;
  %else %do;
    %put ENVIRONMENT NOT SPECIFIED;
    abort abend;
  %end;
```

```
/* if a filename is specified then build the where clause */
%if "&fnm" ne ""  %then %do;
  data _null_;
    length fullwhr $2000.;
    retain fullwhr;

    /* read in each log file and check for undesired messages */
    %let f = 1;
    %let typ = %scan(&fnm, &f, "&delm");

    /* loop through each type of filename to build the where clause */
    /* embed &typ in double quotes in case filename has any special */
    /* characters or spaces                                         */
    %do %while ("&typ" ne "");

      partwhr = catt("index(flog, '", "&typ", "')");
      fullwhr = catx(" or ", fullwhr, partwhr);

      call symputx('fullwhr', fullwhr);

      %let f = %eval(&f + 1);
      %let typ = %scan(&fnm, &f, "&delm");
    %end;

  run;
%end;

/* need to build pipe directory statement as a macro var  */
/* because the statement requires a series of single and  */
/* double quotes - by building the directory statement    */
/* this allows the user to determine the directory rather */
/* than it being hardcoded into the program               */
/* macro var will be of the form:'dir "directory path" '  */
data _null_;
  libnm = "&loc";
  dirnm = catx(" ", "'", "&ppcmd", quote(libnm), "'");
  call symputx('dirnm', dirnm);
run;

/* read in the contents of the directory containing the logs */
filename pdir pipe &dirnm lrecl=32727;

data logs (keep = flog fdat ftim filename numtok);
  infile pdir truncover scanover;
  input filename $char1000.;

  length flog $25 fdat ftim $10;

  /* keep only the logs */
  if index(filename, ".log");

  /* count the number of tokens (i.e., different parts of filename) */
  /* if there are no spaces then there should be 5 tokens for WIN   */
  /* or 9 tokens for LIN X64                                        */
  numtok = countw(filename,' ','q');

  /* need to build the flog value based on number of tokens */
  /* if there are spaces in the log name then need to grab  */
  /* each piece of the log name                             */
  /* the first token that is retrieved will have '.log' and */
  /* it needs to be removed by substituting a blank         */
  /* also need to parse out the date and time these are in  */
```

```
    /* specific spots within the filename so aren't based on  */
    /* number of tokens but will have different locations     */
    /* depending on environment - so parsing of each piece of */
    /* information will be environment dependent              */
    /* note on the scan function a negative # scans from right*/
    /* and a positive # scans from the left                   */

    /*********** WINDOWS ENVIRONMENT ************/
    /* the pipe will read in the information in */
    /* the format of: date time am/pm size file */
    /* e.g. 08/24/2015 09:08 PM 18,498 ae.log   */
    /*    '08/24/2015' is first token from left */
    /*    'ae.log' is first token from right    */
    %if &sysscp = WIN %then %do;
      do j = 5 to numtok;
        tlog = tranwrd(scan(filename, 4 - j, " "),  ".log", "");
        flog = catx(" ", tlog, flog);
      end;
      ftim = catx(" ", scan(filename, 2, " "), scan(filename, 3, " "));
      fdat = put(input(scan(filename, 1, " "), mmddyy10.), date9.);
    %end;

    /***************************** UNIX ENVIRONMENT ****************************/
    /* the pipe will read in the information in the format of: permissions, user,*/
    /* system environment, file size, month, day, year or time, filename         */
    /* e.g. -rw-rw-r-- 1 userid sysenviron 42,341 Oct 22 2015 ad_adaapasi.log    */
    /*    '-rw-rw-r--' is first token from left                                  */
    /*    'ad_adaapasi.log' is first token from right                            */
    %else %if &sysscp = LIN X64 %then %do;
      do j = 9 to numtok;
         tlog = tranwrd(scan(filename, 8 - j, " "),  ".log", "");
         flog = catx(" ", tlog, flog);
      end;
      _ftim = scan(filename, 8, " ");

      /* in Unix if year is current year then time stamp is displayed */
      /* otherwise the year last modified is displayed                */
      /* so if no year is provided then default to today's year and if*/
      /* no time is provided indicated 'N/A'                          */
      if anypunct(_ftim) then do;
        ftim = put(input(_ftim, time5.), timeampm8.);
        yr = put(year(today()), Z4.);
      end;
      else do;
        ftim = 'N/A';
        yr = _ftim;
      end;

      fdat = cats(scan(filename, 7, " "), upcase(scan(filename, 6, " ")), yr);
    %end;
  run;

  /* create a list of logs, dates, times and store in macro variables */
  proc sql noprint;
    select flog,
           fdat,
           ftim
           into : currlogs separated by "&delm",
                     : currdats separated by " ",
                          : currtims separated by "@"
    from logs
    %if "&fnm" ne "" %then where &fullwhr;     ; /* need to keep extra semicolon */
  quit;
```

```
/* need to make sure the alllogs data set does not exist before getting into loop */
proc datasets;
   delete alllogs;
quit;

/* read in each log file and check for undesired messages */
%let x = 1;
%let lg = %scan(&currlogs, &x, "&delm");
%let dt = %scan(&currdats, &x);
%let tm = %scan(&currtims, &x, '@');

/* loop through each log in the directory and look for undesirable messages */
/* embed &lg in double quotes in case filename has special characters/spaces*/
%do %while ("&lg" ne "");
  /* read the log file into a SAS data set to parse the text */
  data logck&x;
    infile "&loc.&slash.&lg..log" missover pad;
    input line $1000.;

    /* keep only the records that had an undesirable message */
    if index(upcase(line), "WARNING") or
       index(upcase(line), "ERROR:") or
       index(upcase(line), "UNINITIALIZED") or
       index(upcase(line), "NOTE: MERGE") or
       index(upcase(line), "MORE THAN ONE DATA SET WITH REPEATS OF BY") or
       index(upcase(line), "VALUES HAVE BEEN CONVERTED") or
       index(upcase(line), "MISSING VALUES WERE GENERATED AS A RESULT") or
       index(upcase(line), "INVALID DATA") or
       index(upcase(line), "INVALID NUMERIC DATA") or
       index(upcase(line), "AT LEAST ONE W.D FORMAT TOO SMALL") or
       index(upcase(line), "ORDERING BY AN ITEM THAT DOESN'T APPEAR IN") or
       index(upcase(line), "OUTSIDE THE AXIS RANGE") or
       index(upcase(line), "RETURNING PREMATURELY") or
       index(upcase(line), "UNKNOWN MONTH FOR") or
       index(upcase(line), "QUERY DATA") or
       index(upcase(line), "??") or
       index(upcase(line), "QUESTIONABLE");

    /* create variables that will contain the log that is being scanned */
    /* as well as the and date and time that the log file was created   */
    length lognm $25. logdt logtm $10.;
    lognm = upcase("&lg");
    logdt = "&dt";
    logtm = "&tm";

    /* create a dummy variable to create a column on the report that will allow */
    /* users to enter a reason if the message is allowed                        */
    logrs = ' ';
  run;

  /* because there are sometimes issues with SAS certificate */
  /* there will be warnings in the logs that are expected    */
  /* these need to be removed                                */
  data logck&x._2;
    set logck&x.;
    if index(upcase(line), 'UNABLE TO COPY SASUSER') or
       index(upcase(line), 'BASE PRODUCT PRODUCT') or
       index(upcase(line), 'EXPIRE WITHIN') or
       (index(upcase(line), 'BASE SAS SOFTWARE') and
        index(upcase(line), 'EXPIRING SOON')) or
       index(upcase(line), 'UPCOMING EXPIRATION') or
       index(upcase(line), 'SCHEDULED TO EXPIRE') or
```

```
      index(upcase(line), 'SETINIT TO OBTAIN MORE INFO') then delete;
  run;

  /* determine the number of undesired messages were in the log */
  data _null_;
    if 0 then set logck&x._2 nobs=final;
    call symputx('numobs',left(put(final, 8.)));
  run;

  /* if there is no undesired messages in log create a dummy record for report */
  %if &numobs = 0 %then %do;
    data logck&x._2;
      length lognm $25. line $1000. logdt logtm $10.;
      line = "No undesired messages.  Log is clean.";
      lognm = upcase("&lg");
      logdt = "&dt";
      logtm = "&tm";

      /* create a dummy variable to create a column on the report that will allow */
      /* users to enter a reason if the message is allowed                        */
      logrs = ' ';
      output;
    run;
  %end;

  /* append all the results into one data set */
  %if &x = 1 %then %do;
    data alllogs;
      set logck&x._2;
    run;
  %end;
  %else %do;
    proc append base=alllogs
                new=logck&x._2;
    run;
  %end;

  %let x = %eval(&x + 1);
  %let lg = %scan(&currlogs, &x, "&delm");
  %let dt = %scan(&currdats, &x);
  %let tm = %scan(&currtims, &x, '@');
%end;

/* since a list of files can be provided then the files may not be in order */
proc sort data=alllogs presorted;
  by lognm line;
run;

/* if the name of the output file is not specified then default to the name */
%if "&out" =  "" %then %do;
  %let out=all_checklogs;
%end;

/* if the name of the output file is not specified then default to the name */
%if "&loc2" = "" %then %do;
  data _null_;
       call symputx("loc2", "&loc");
    run;
%end;

/* create the report */
ods listing close;
options orientation=landscape;
```

```
  ods rtf file="&loc2.&slash.&out..rtf";

  proc report data=alllogs ls=140 ps=43 spacing=1 missing nowindows headline;
    column lognm logdt logtm line logrs;
    define lognm / order   style(column)=[width=12%]      "Log Name";
    define logdt / display style(column)=[width=12%]      "Log Date";
    define logtm / display style(column)=[width=12%]      "Log Time";
    define line  / display style(column)=[width=30%] flow "Log Message";
    define logrs / display style(column)=[width=20%] flow "Reason Message is Allowed";

    /* force a blank line after each file */
    compute after lognm;
      line " ";
    endcomp;
  run;

  ods rtf close;
  ods listing;
%mend checklogs;
```