SAS® Debugging 101

Kirk Paul Lafler, Software Intelligence Corporation, Spring Valley, California

Abstract

SAS® users are almost always surprised to discover their programs contain bugs. In fact, when asked users will emphatically stand by their programs and logic by saying they are bug free. But, the vast number of experiences along with the realities of writing code says otherwise. Bugs in software can appear anywhere; whether accidentally built into the software by developers, or introduced by programmers when writing code. No matter where the origins of bugs occur, the one thing that all SAS users know is that debugging SAS program errors and warnings can be a daunting, and humbling, task. This presentation explores the world of SAS bugs, providing essential information about the types of bugs, how bugs are created, the symptoms of bugs, and how to locate bugs. Attendees learn how to apply effective techniques to better understand, identify, and repair bugs and enable program code to work as intended.

Introduction

From the very beginning of computing history, program code, or the instructions used to tell a computer how and when to do something, has been plagued with errors, or bugs. Even if your own program code is determined to be error-free, there is a high probability that the operating system and/or software being used, such as the SAS software, have embedded program errors in them. As a result, program code should be written to assume the unexpected and, as much as possible, be able to handle unforeseen events using acceptable strategies and programming methodologies.

The focus of this paper is to understand the various types of program errors and their causes, in order to achieve greater success when investigating and identifying program errors before and when they occur. By accomplishing this, users expect to improve their chances of preventing, fixing and/or removing program errors the best way possible.

The SAS Log – Notes, Warnings and Errors

Users are provided with a broad range of assistance while using the various services offered by the SAS system, including the display of SAS log messages related to compile-time and execution-time scenarios. The types of SAS log messages include notes, warnings and errors, where the specific notes, warnings and errors relate to the degree of success (or lack thereof) when applying the rules of the language, to specific data types, and to the creation and use of macro variables. To help in understanding the various types of program errors found in the SAS System, a classification system is applied along with a brief description, as illustrated in Table 1.

The type and severity of error encountered depends on the specific application being worked on. The SAS log displays violations related to syntax, semantic, execution-time, data and macro-related errors. The assortment of notes, warnings and errors provide users with helpful information to better understand and, hopefully, debug DATA, PROC and macro step program code.

Type of Error	Description
Syntax Error	The rules associated with the use of a programming statement in the SAS language are violated. Typical examples of a syntax error include misspelling of dataset or variable names, and forgetting a semicolon.
Semantic Error	An element in a statement is specified incorrectly preventing SAS from knowing how to interpret your code. Typical examples of semantic errors include misspelling a variable name and incorrectly specifying an array's elements.
Execution-time Error	An error that produces a Note or Warning on the SAS Log, but allows the program to continue. An example includes observations arranged in incorrect BY-group order.
Data Error	An error that is generated when one or more data values do not match an INPUT statement specification.
Macro-Related Error	An error that occurs during macro compilation or execution. An example includes a local macro variable that should have been defined as a global macro variable.
Logic Error	An error that does not have a Note, Warning or Error associated with it, but contains erroneous or unexpected results. Examples include using a "<" when a ">" comparison operator should have been specified.

Table 1. Error Classification and Description

Understanding Where Error Processing Occurs

In the SAS System, error processing occurs during the compilation and execution phases of SAS processing. As illustrated in Table 2, of the six types of errors, SAS recognizes five of them either at compile or execution time.

Errors During Compilation Phase	Errors During Execution Phase
Syntax Error	Execution-time Error
Semantic Error	Data Error
Macro-Related Error	Macro-Related Error

Table 2. Types of Errors and Error Processing Phases

Difficult to Find Errors

Users, with the aid of the SAS Log, are able to determine the cause of many types of errors, but there are situations where SAS is able to offer very little help with – Logic (or "Usage") errors. A good strategy to use in the detection and resolution of Usage errors is to first develop a baseline understanding of what the program code is expected to do and then carefully compare these expectations to the output that was generated. To detect Usage errors in a DATA step, users can use the DATA Step Debugger (for more information, see Debugging SAS® Programs, by Michelle Burlew, pps. 110-138).

Exploring SAS System Options

Users are able to examine their current SAS System options, along with the current default settings by using one of the following approaches:

1. Using the PROC OPTIONS statement

   ```
   PROC OPTIONS;
   RUN;
   ```

2. Viewing the OPTIONS window

3. Accessing the contents of DICTIONARY.OPTIONS using the SQL procedure

   ```
   PROC SQL;
      SELECT * FROM DICTIONARY.OPTIONS;
   QUIT;
   ```

4. Accessing the virtual table SASHELP.VOPTION using any procedure and/or DATA step

   ```
   PROC PRINT DATA=SASHELP.VOPTION;
   RUN;
   ```

DATA and PROC Step Debugging with System Options

Using one or more SAS System options will allow users to:

- Process information to be displayed in the SAS Log;
- Run programs with specific observations;
- Stop a program when certain errors are encountered.

SAS System options can be specified singly, or in combination, depending on needs. A number of SAS System options will now be presented and illustrated to show their debugging capabilities.

Using DSNFERR / NODSNFERR System Option

The DSNFERR system option can be used to tell SAS to stop processing when a reference to a data set does not exist.

Example:

```
options  DSNFERR;
 data libref.movies;
   set work.movies;
   <  other SAS statements  >;
 run;
```

Using ERRORABEND / NOERRORABEND System Option

The ERRORABEND system option can be used to tell SAS to abnormally terminate for many errors including syntax errors.

Example:

```
options  ERRORABEND;
 data libref.movies;
   set work.movies;
   <  other SAS statements  >;
 run;
```

Using ERRORS= n System Option (default is 20)

The ERRORS= n system option can be used to tell SAS the maximum number of observations to print complete error messages for.

Example:

```
options  ERROR=100;
 data libref.movies;
   set work.movies;
   <  other SAS statements  >;
 run;
```

Using FMTERR / NOFMTERR System Option

The FMTERR system option can be used to tell SAS to produce an error when it cannot find a format.

Example:

```
options  FMTERR;
 data libref.movies;
   set work.movies;
   <  other SAS statements  >;
 run;
```

Using MSGLEVEL= N / I System Option

The MSGLEVEL=I system option can be used to tell SAS to print notes, warnings, errors and informational messages for merge, index and sort usage.

Example:

```
options  MSGLEVEL=I;
 data libref.movies;
   set work.movies;
   <  other SAS statements  >;
 run;
```

Using NOTES / NONOTES System Option

The NOTES system option can be specified to tell SAS to print all notes to the SAS Log.

Example:

```
options NOTES;
 data libref.movies;
   set work.movies;
   < other SAS statements >;
 run;
```

Using DATASTMTCHK=COREKEYWORDS System Option

The DATASTMTCHK=COREKEYWORDS system option can be used to prevent a data set from being overwritten when there is a syntax error in a MERGE, SET, UPDATE or MODIFY statement.

Example:

```
options DATASTMTCHK=COREKEYWORDS;
 data libref.movies;
   set work.movies;
   < other SAS statements >;
 run;
```

Using REPLACE / NOREPLACE System Option

The NOREPLACE system option can be specified to prevent the replacement of permanent data sets.

Example:

```
options NOREPLACE;
 data libref.movies;
   set work.movies;
   < other SAS statements >;
 run;
```

Using SOURCE / NOSOURCE System Option

The SOURCE system option can be used to write all source code to the SAS Log.

Example:

```
options SOURCE;
 data libref.movies;
   set work.movies;
   < other SAS statements >;
 run;
```

Using SOURCE2 / NOSOURCE2 System Option

The SOURCE2 system option can be specified to write all included source code (used with the %include statement) to the SAS Log.

Example:

```
options SOURCE2;
 data libref.movies;
   set work.movies;
  %include 'c:\include-sas-code.sas';
   < other SAS statements >;
 run;
```

Using OBS=0 and NOREPLACE System Options

This combination of options tells SAS to execute the step and analyze the syntax of your code without reading any input data.

Example:

```
options OBS=0 NOREPLACE;
 data libref.movies;
   set work.movies;
   < other SAS statements >;
 run;
```

Macro Debugging Strategies and Techniques

A number of macro debugging strategies and techniques exist. But, before assuming an error is macro-oriented, a general litmus test can be applied to first determine whether the error is a macro-related or a non-macro problem.

- If the message displays a number, then the error is most likely due to non-macro SAS code;
- Otherwise, the error can be assumed to be a macro-related error.

Isolating Macro Issues

It's often useful to isolate macro issues by displaying their value after macro resolution to help determine whether problems exist. One approach used by macro enthusiasts to isolate macro issues is to:

- Specify the %PUT statement to isolate problems;
- Display the contents after macro resolution to the SAS log;
- Inspect and verify:
  - ✓ a macro variable's value;
  - ✓ ampersand resolution;
  - ✓ a specified condition was met;
  - ✓ leading or trailing blanks in a value.

Other strategies and techniques exist for isolating macro issues. The following approaches are presented to illustrate techniques that the author has effectively used.

MLOGIC and Flow of Execution

The MLOGIC option is an effective debugging tool that is used to trace the macro's execution writing the results of the trace information to the SAS log. The following example code illustrates using the MLOGIC option.

SAS Code

```
OPTIONS  MLOGIC;
%MACRO statsproc (PROC, DSN);
  %IF %UPCASE(&proc)=MEANS %THEN %DO;
          PROC MEANS DATA=&dsn;
          RUN;
  %END;
  %ELSE %DO;
          PROC UNIVARIATE DATA=&dsn;
          RUN;
  %END;
%MEND statsproc;
%statsproc (means, SASUSER.movies);
```

SAS Log Results

```
1   OPTIONS MLOGIC;
2   %MACRO statsproc (PROC, DSN);
3     %IF %UPCASE(&proc)=MEANS %THEN %DO;
4         PROC MEANS DATA=&dsn; RUN;
5     %END;
6     %ELSE %DO;
7         PROC UNIVARIATE DATA=&dsn; RUN;
8     %END;
9   %MEND statsproc;
10  %statsproc (means, SASUSER.movies);
MLOGIC(STATSPROC):  Beginning execution.
MLOGIC(STATSPROC):  Parameter PROC has value means
MLOGIC(STATSPROC):  Parameter DSN has value SASUSER.movies
MLOGIC(STATSPROC):  %IF condition %UPCASE(&proc)=MEANS is TRUE
NOTE: There were 22 observations read from the data set SASUSER.MOVIES.
NOTE: PROCEDURE MEANS used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
MLOGIC(STATSPROC):  Ending execution.
```

MPRINT and Generated SAS Code

The MPRINT option is an effective debugging technique to display the SAS statements that have been generated by macro execution. The results of the MPRINT option are written to the SAS log. The following example code illustrates using the MPRINT option.

SAS Code

```
OPTIONS  MPRINT;
%MACRO statsproc (PROC, DSN);
  %IF %UPCASE(&proc)=MEANS %THEN %DO;
          PROC MEANS DATA=&dsn;
          RUN;
  %END;
  %ELSE %DO;
          PROC UNIVARIATE DATA=&dsn;
          RUN;
  %END;
%MEND statsproc;
%statsproc (means, SASUSER.movies);
```

SAS Log Results

```
1    OPTIONS MPRINT;
2    %MACRO statsproc (PROC, DSN);
3      %IF %UPCASE(&proc)=MEANS %THEN %DO;
4          PROC MEANS DATA=&dsn; RUN;
5      %END;
6      %ELSE %DO;
7          PROC UNIVARIATE DATA=&dsn; RUN;
8      %END;
9    %MEND statsproc;
10    %statsproc (means, SASUSER.movies);
MPRINT(STATSPROC):   PROC MEANS DATA=SASUSER.movies;
MPRINT(STATSPROC):   RUN;
NOTE: There were 22 observations read from the data set SASUSER.MOVIES.
NOTE: PROCEDURE MEANS used (Total process time):
      real time           0.00 seconds
      cpu time            0.01 seconds.
```

Debugging with Benchworkzz® Software

Users who have written and executed a SAS program of DATA, PROC and/or macro code are all too familiar with the sometimes tedious and time-consuming process of debugging a program using the valuable content displayed in the SAS log. The typical debugging approach has users starting at the top of the log, carefully inspecting, one by one, each helpful note, warning, error and informative message. This approach not only provides a better understanding of how a program works in relation to its intended design, but offers greater insight in how SAS works. For many SAS users, the process of examining the log for notes, warnings, errors and informative messages is a normal course of action, while others look to leverage this time-consuming process by using a software tool. One such tool, Benchworkzz, developed by Benchworkzz, LLC in Houston, Texas, is designed to provide users with an easy-to-use productivity and diagnostic tool to help discover all those SAS log NOTE, WARNING, ERROR, and INFO messages. By using the Benchworkzz debugging tool, users have an all-in-one tool to help scrub and sanitize program code systematically and economically.

The Benchworkzz Application Interface

Once the Benchworkzz tool is launched, you immediately see the application interface, illustrated in Figure 1. The Benchworkzz tool is particularly useful for simplifying the process of finding any, and all, SAS log NOTE, WARNING, ERROR, and INFO messages produced during the compilation and execution phases of SAS processing.

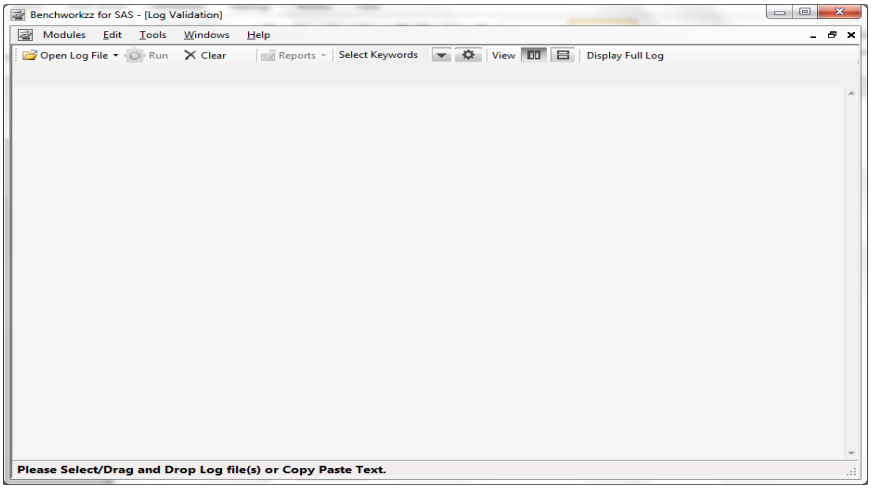

Figure 1. The Benchworkzz Application Interface

Benchworkzz Navigation and Drop-down Menus

The process of navigation within Benchworkzz is simplified with an assortment of built-in drop-down menus: Modules, Edit, Tools, Windows and Help, as illustrated in Figure 2. The Modules drop-down menu provides users with the ability to access the Log Validation module or the Text Parser module (each module will be discussed separately later in this paper). The Edit drop-down menu offers users with the familiar assortment of editing features, such as Undo, Redo, Cut, Copy, Paste and Select All. The Tools drop-down menu provides users with features such as Launch on Startup, Shortcut On Desktop, Ftp Settings and Options. The Windows drop-down menu offers users the ability to Cascade, Tile Vertically, Tile Horizontally, Close All and Arrange Icons. Finally, the Help drop-down menu provides users with Help Topics, BenchworkzzGlobal.com, About Benchworkzz for SAS and Check for Updates.

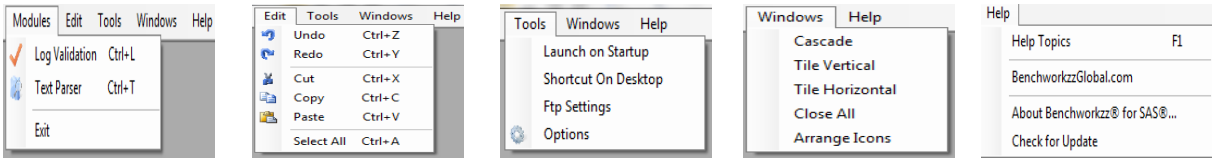

Figure 2. Benchworkzz Drop-down Menus

Using the Benchworkzz Log Validation Module

The Log Validation module provides users with a powerful search engine that helps filter specific NOTE, WARNING, ERROR and INFO log status events, see Figure 3.

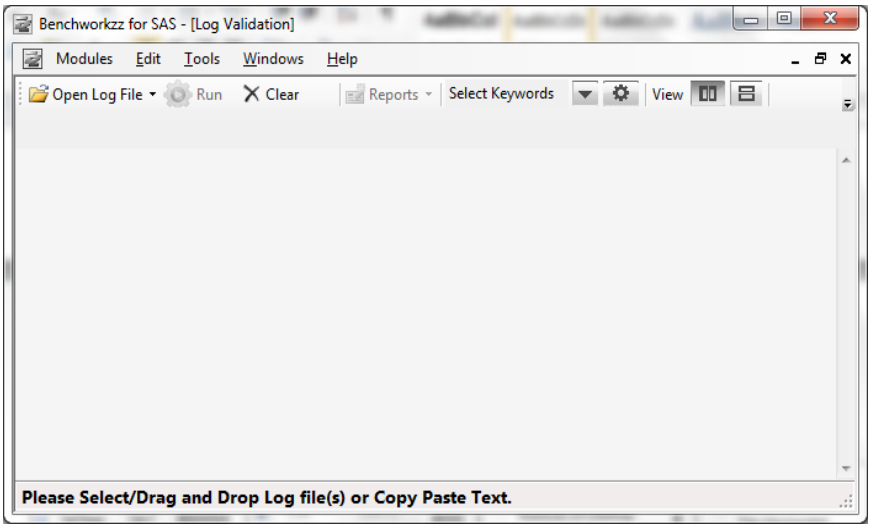

Figure 3. Benchworkzz Log Validation Module

The Log Validation Module Drop-down Menus

Users are provided with a number of powerful Log Validation Module drop-down menus for opening Log input files, a repository of master reference keywords that represents serious errors reported in the log, and a repository containing user-supplied optional keywords for less severe keywords that can be ignored during processing, as shown in Figure 4.

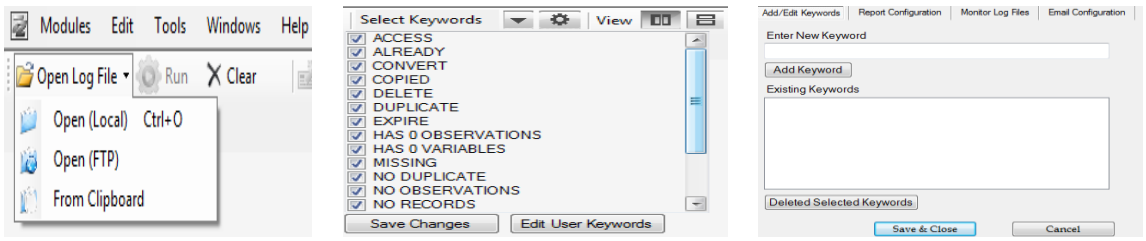

Figure 4. Log Validation Module Drop-down Menus

To illustrate the workings of the Log Validation Module, a simple DATA step hash object code (containing a number of intentional errors) was submitted in the SAS System, shown in Figure 5.

SAS Code

```
data _null_;

  if 0 then set mydata.actors;  /* load variable properties into hash tables */
  if _n_ = 1 then do;
     declare Hash MatchTitles (dataset:'mydata.actors'); /* declare the name MatchTitles
                                                            for hash */

     MatchTitles.DefineKey ('Title'); /* identify variable to use as key */
     MatchTitles.DefineData ('Actor_Leading',
                             'Actor_Supporting'); /* identify columns of data */
     MatchTitles.DefineDone (); /* complete hash table definition */
  end;

  set mydata.movies end=eof;

  if eof and
     MatchTitles.find(key:title) = 0 then   /* write data using hash MatchTitles */
          MatchTitles.output(dataset:match_on_titles);
run;
```

Figure 5. DATA Step Hash Object Code (with errors)

After the DATA step hash object code was executed, the contents of the SAS log was saved to a folder on my hard drive. By clicking the Open Log File drop-down menu the Please Select a SAS Log File window appears allowing navigation to the folder where the SAS log contents was saved, as shown in Figure 6. The Run button was then clicked to run the log validator

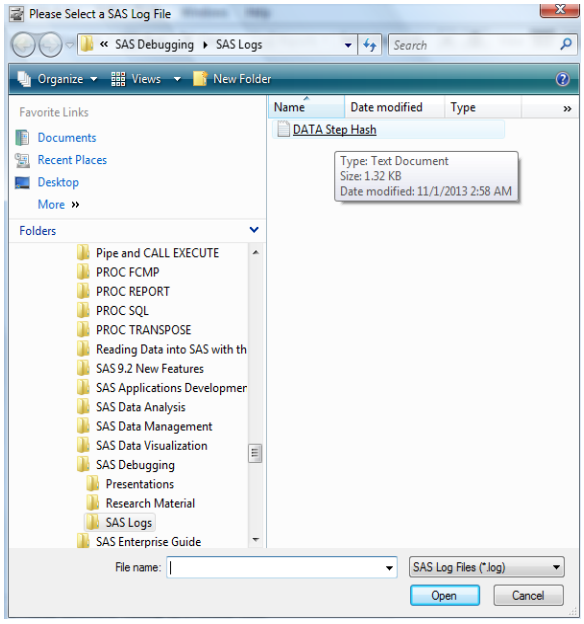

Figure 6. Please Select a SAS Log File Window

With the SAS log file selected, the Log Validation Module is ready to process the file. After clicking the Run button to initiate the log validator, the results from the log validation process are displayed, as shown in Figure 7. The results show 25 Log lines were processed by the Log Validation process with 4 events found: 3 ERRORS and 1 NOTE.

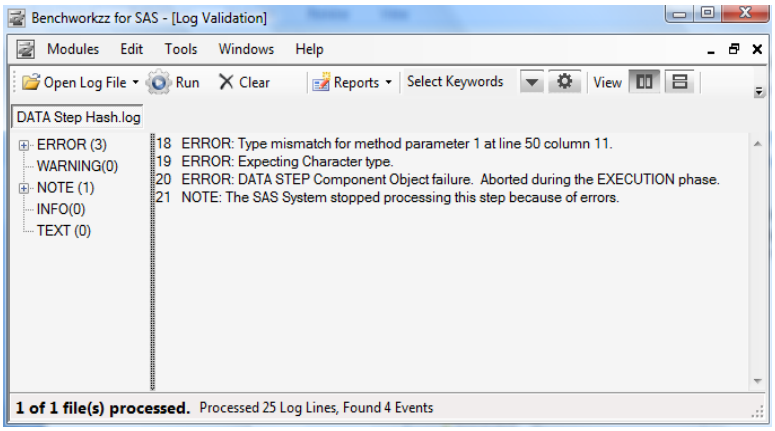

Figure 7. Log Validation Results

Users are able to produce log validation reports as well. By clicking the Reports drop-down menu, users are able to view (and print) a summary report, as shown in Figure 8, or a detailed report, as shown in Figure 9.

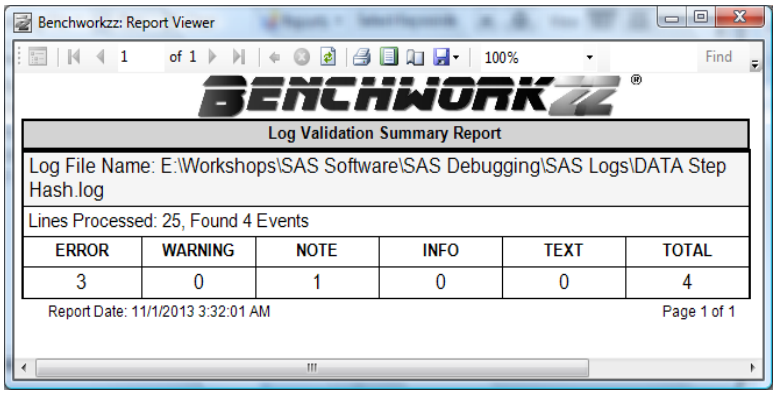

Figure 8. Log Validation Summary Report

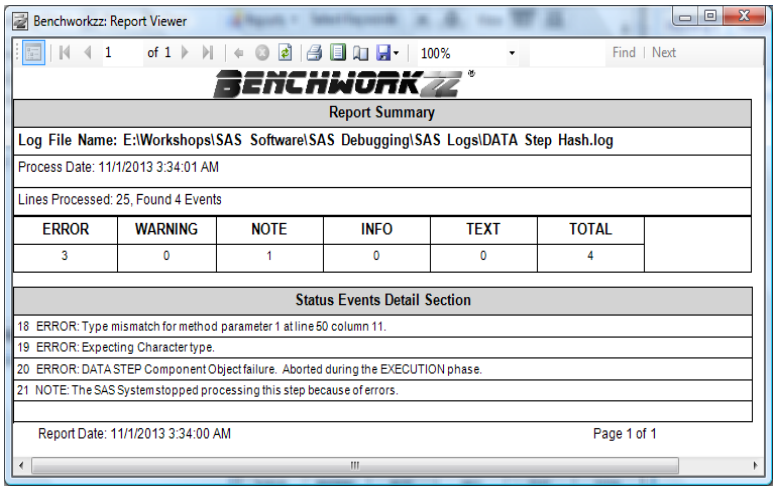

Figure 9. Log Validation Detailed Report

Using the Benchworkzz Text Parser Module

The Benchworkzz Text Parser provides users with the ability to search SAS Log files, as well as any text file, for a list of phrases and specific terms; text matching, wildcard and regular expressions, as illustrated in Figure 10.

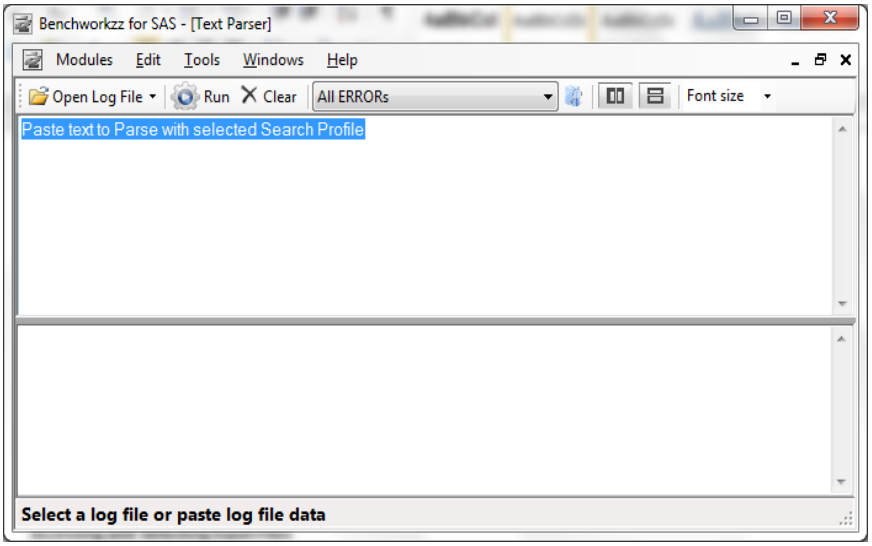

Figure 10. Benchworkzz Text Parser Module

To illustrate how the Benchworkzz Text Parser Module works, the DATA step hash object code we used earlier with the Log Validation Module was executed and the SAS log results copied from the log window and pasted into the Text Parser Module. With the default setting, All ERRORs, selected only ERROR-type messages will be captured and displayed in the lower panel of the Text Parser window, as shown in Figure 11.

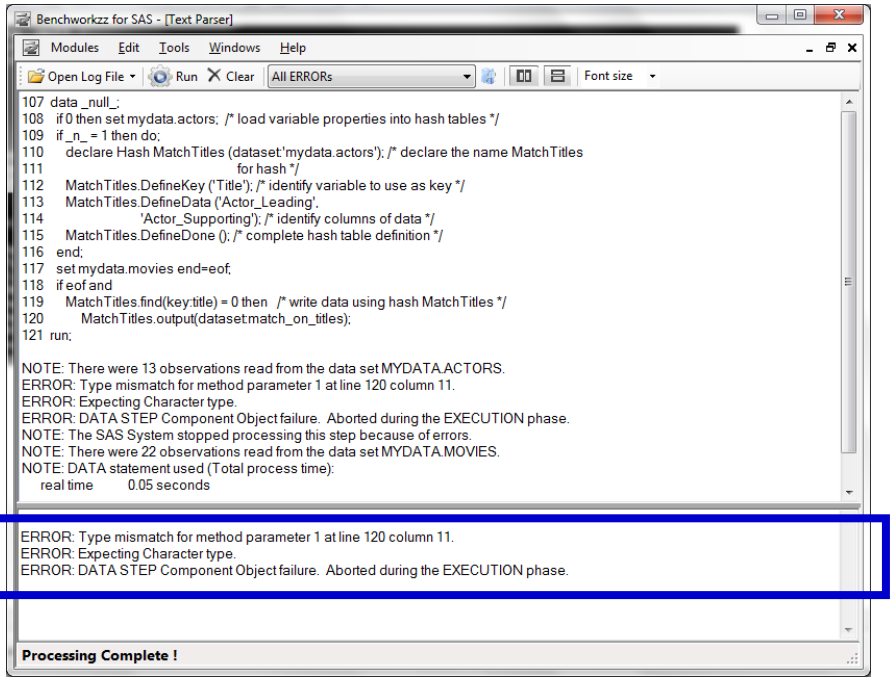

Figure 11. Errors Displayed in the Text Parser Module

Search Profiler Editor

Users have access to a powerful editor, the Search Profiler Editor, to assist with finding specific text within the log or file, a text replacement feature where selected text can be replaced with some other text, a text matching process for finding text that starts and/or ends with specific text, an exclusion feature to exclude specific text, and a user-supplied repository where search keywords and/or phrases can be entered, as shown in Figure 12.

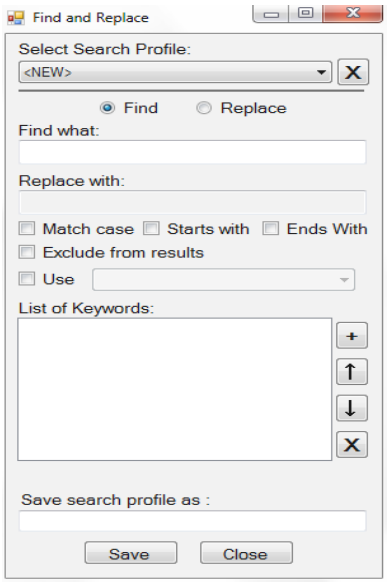

Figure 12. Search Profiler Editor - Find and Replace Dialog Box

Conclusion

The focus of this paper has been to understand the various types of program errors along with their causes, in order to achieve greater success in investigating and identifying program errors before and when they occur. Various strategies and techniques were illustrated to debug SAS program errors and warnings, including using system options to debug DATA and PROC step code, system options to debug macro code, return codes to assist the debugging process, and a brief introduction to Benchworkzz software to understand, fix, and resolve errors and warnings. The expectations are to improve our chances of preventing, fixing and/or resolving program errors allowing code to work as intended.

References

Benchworkzz® for SAS® User's Guide; Benchworkzz, Houston, Texas, USA.

Brin, Sergey and Lawrence Page (1998), *"The Anatomy of a Large-Scale Hypertextual Web Search Engine,"* Computer Science Department, Stanford University, Stanford, California, http://infolab.stanford.edu/~backrub/google.html.

Burlew, M., Debugging SAS® Programs: A Handbook of Tools and Techniques, SAS Publishing.

Delwiche, Lora d. and Susan J. Slaughter (2003); *"Errors, Warnings and Notes (Oh My) A Practical Guide to Debugging SAS® Programs,"* Proceedings of the 2003 SAS Users Group International (SUGI) Conference.

DiIorio, Frank., The SAS(r) Debugging Primer, http://www2.sas.com/proceedings/sugi26/p054-26.pdf.

Fahmy, Adel (2010); *"Logging the Log Magic: Pulling the Rabbit out of the Hat,"* Proceedings of the 2010 PharmaSUG Conference; Benchworkzz, Austin, Texas, USA.

Lafler, Kirk Paul (2014); *"SAS® Debugging 101,"* Proceedings of the 2014 SouthEast SAS Users Group (SESUG) Conference; Software Intelligence Corporation, Spring Valley, California, USA.

Lafler, Kirk Paul (2014); *"SAS® Debugging 101,"* Proceedings of the 2014 MidWest SAS Users Group (MWSUG) Conference; Software Intelligence Corporation, Spring Valley, California, USA.

Lafler, Kirk Paul (2014); *"Strategies and Techniques for Debugging SAS® Program Errors and Warnings,"* Proceedings of the 2014 PharmaSUG Conference; Software Intelligence Corporation, Spring Valley, California, USA.

Lafler, Kirk Paul (2013); *"Strategies and Techniques for Debugging SAS® Program Errors and Warnings,"* Proceedings of the 2013 MidWest SAS Users Group (MWSUG) Conference; Software Intelligence Corporation, Spring Valley, California, USA.

Russell, Kevin and Russ Tyndall (2010); *"SAS® System Options: The True Heroes of Macro Debugging,"* Proceedings of the 2010 NorthEast SAS Users Group (NESUG) Conference; SAS Institute Inc., Cary, North Carolina, USA.

SAS Institute Inc., SAS® Language Reference: Concepts, Cary, NC: SAS Institute Inc., 1999. 554 pages.

Staum, R., To Err is Human; to Debug, Devine, http://www2.sas.com/proceedings/sugi27/p064-27.pdf.

Acknowledgments

The author thanks Alejandro Farias and John Taylor, SCSUG 2014 Conference Chairs, for accepting my abstract and paper; as well as SAS Institute Inc.; and the SCSUG Executive Committee for organizing a great conference!

Trademark Citations

About the Author

Kirk Paul Lafler is consultant and founder of Software Intelligence Corporation and has been using SAS since 1979. He is a SAS Certified Professional, provider of IT consulting services, trainer to SAS users around the world, and sasCommunity.org emeritus Advisory Board member. As the author of six books including Google® Search Complete (Odyssey Press. 2014); PROC SQL: Beyond the Basics Using SAS, Second Edition (SAS Press. 2013); PROC SQL: Beyond the Basics Using SAS (SAS Press. 2004); Kirk has written more than five hundred papers and articles, been an Invited speaker and trainer at four hundred-plus SAS International, regional, special-interest, local, and in-house user group conferences and meetings, and is the recipient of 23 "Best" contributed paper, hands-on workshop (HOW), and poster awards.

Comments and suggestions can be sent to:

Kirk Paul Lafler
Senior SAS® Consultant, Application Developer, Data Scientist, Trainer and Author
Software Intelligence Corporation
E-mail: KirkLafler@cs.com
LinkedIn: http://www.linkedin.com/in/KirkPaulLafler
Twitter: @sasNerd