PharmaSUG 2017 - Paper TT03

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 and Fix Errors

With the aid of the SAS Log, users are better able to determine the cause of many types of errors, but there is one areas where SAS offers very little help – 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).

In the paper, "How to find and fix programming errors", by Rick Wicklin, Rick suggests the following guidelines to help find and fix those pesky programming errors.

- Test the program on simple cases;
- Break down the program into a sequence of basic steps;
- Favor clarity and simplicity.

Exploring SAS System Options

SAS System options offer users with a wealth of power to manage input, output, processing, performance, communication, log and procedure output control, sort, macro and other essential elements related to a SAS session. Users are able to examine their current SAS System options, along with the current default settings by using one of the following approaches:

1. Viewing an alphabetical list of system options and their default settings using the PROC OPTIONS statement:

```
PROC OPTIONS;
RUN;
```

2. Viewing the contents of the system options and their default settings using the **OPTIONS** Display Manager command.

3. Accessing the contents of the metadata **DICTIONARY.OPTIONS** table using the SQL procedure:

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

4. Accessing the contents of the virtual table (view) **SASHELP.VOPTION** using any procedure and/or DATA step:

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

DATA and PROC Step Debugging with System Options

SAS System options can be specified singly, or in combination, depending on needs. SAS System options 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.

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.

<u>Example:</u>

```
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.

<u>Example:</u>

```
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.

<u>Example:</u>

```
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.

<u>Example:</u>

```
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.

<u>Example:</u>

```
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-generated, a general litmus test can be applied to determine whether the error is a macro-related or a non-macro problem. Inspect the message displayed in the SAS log and:

- 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.

Although this two-step rule can be a helpful technique much of the time, it is worth noting that it can become less than reliable when a programmer creates their own error numbering scheme. In these cases, the debugging process may require the use of other techniques, including isolating macro issues by specifying a %PUT statement.

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 techniques are presented to illustrate the approaches the author has successfully used to better understand and debug macro code.

SYMBOLGEN and the Resolution of Macro Variables

The SYMBOLGEN option is an effective debugging tool used to display the resolution of each macro variable. The following example code illustrates using the SYMBOLGEN option.

SAS Code

```
OPTIONS SYMBOLGEN;
%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

```
108  OPTIONS SYMBOLGEN;
109  %MACRO statsproc (PROC, DSN);
110    %IF %UPCASE(&proc)=MEANS %THEN %DO;
111            PROC MEANS DATA=&dsn;
112            RUN;
113    %END;
114    %ELSE %DO;
115            PROC UNIVARIATE DATA=&dsn;
116            RUN;
117    %END;
118  %MEND statsproc;
119  %statsproc (means, SASUSER.movies);

SYMBOLGEN:  Macro variable PROC resolves to means
SYMBOLGEN:  Macro variable DSN resolves to SASUSER.movies
```

MLOGIC and Flow of Execution

The MLOGIC option is an effective debugging tool used to trace the macro's execution writing the results of the trace information to the SAS log. Used most often when dealing with nested macros, it marks the beginning and end of macro execution, displays the values of macro parameters at invocation, displays the execution of each macro program statement, displays whether a %IF logic condition resolves to a "TRUE" or "FALSE" value on the SAS Log, and shows the location of an invoked AUTOCALL macro. The following example code illustrates using the MLOGIC option.

<u>**SAS Code**</u>

```
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);
```

<u>**SAS Log Results**</u>

```
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.
```

MFILE and Finding Macro Errors

The MFILE option is an effective debugging technique that helps to display the exact location of the macro error. The results of the MFILE option are written to the SAS log. The following example code illustrates using the MFILE option for debugging purposes.

SAS Code

```
OPTIONS MFILE;
%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

```
24   OPTIONS MFILE;
25   %MACRO statsproc (PROC, DSN);
26     %IF %UPCASE(&proc)=MEANS %THEN %DO;
27             PROC MEANS DATA=&dsn;
28             RUN;
29     %END;
30     %ELSE %DO;
31             PROC UNIVARIATE DATA=&dsn;
32             RUN;
33   %MEND statsproc;
ERROR: There were 1 unclosed %DO statements.  The macro STATSPROC will not be
compiled.
ERROR: A dummy macro will be compiled.
34   %statsproc (means, SASUSER.movies);
     -
     180
WARNING: Apparent invocation of macro STATSPROC not resolved.

ERROR 180-322: Statement is not valid or it is used out of proper order.
```

MLOGICNEST and the Nesting Sequence of Macros

The MLOGICNEST option is an effective way to display the nesting sequence of macros. Used with the MLOGIC option, the results of the MLOGICNEST option display the outer-most macro to the inner-most macro to better understand the macro nesting structured. The macro code and SAS log are illustrated in the example, below.

SAS Code

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

SAS Log Results

```
71   OPTIONS MLOGIC MLOGICNEST;
72   %MACRO means;
73      PROC MEANS DATA=&dsn;
74      RUN;
75      %DONE;
76   %MEND means;
77   %MACRO univariate;
78      PROC UNIVARIATE DATA=&dsn;
79      RUN;
80      %DONE;
81   %MEND univariate;
82   %MACRO done;
83      %put PROC is finished.;
84   %MEND done;
85   %MACRO statsproc (PROC, DSN);
86     %IF %UPCASE(&proc)=MEANS %THEN %DO;
```

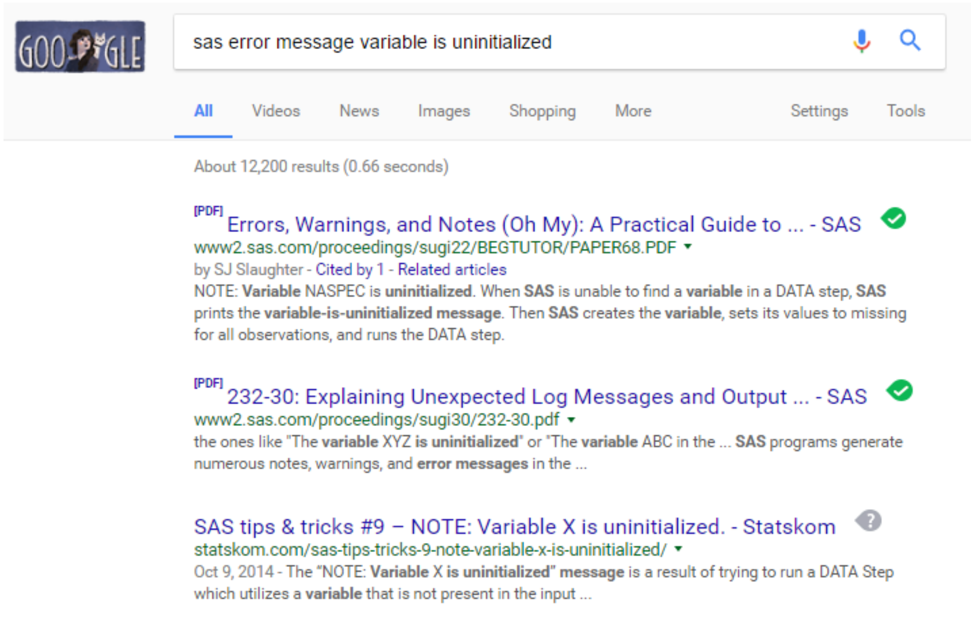
```
87              %MEANS;
88      %END;
89      %ELSE %DO;
90              %UNIVARIATE;
91      %END;
92    %MEND statsproc;
93    %statsproc (means, SASUSER.movies);
MLOGIC(STATSPROC):  Beginning execution.
MLOGIC(STATSPROC):  Parameter PROC has value means
MLOGIC(STATSPROC):  Parameter DSN has value SASUSER.movies
MLOGIC(STATSPROC):  %IF condition %UPCASE(&proc)=MEANS is TRUE
MLOGIC(STATSPROC.MEANS):  Beginning execution.

NOTE: Writing HTML Body file: sashtml2.htm
NOTE: There were 22 observations read from the data set SASUSER.MOVIES.
NOTE: PROCEDURE MEANS used (Total process time):
      real time           0.68 seconds
      cpu time            0.43 seconds


MLOGIC(STATSPROC.MEANS.DONE):  Beginning execution.
MLOGIC(STATSPROC.MEANS.DONE):  %PUT PROC is finished.
PROC is finished.
MLOGIC(STATSPROC.MEANS.DONE):  Ending execution.
MLOGIC(STATSPROC.MEANS):  Ending execution.
MLOGIC(STATSPROC):  Ending execution.
```

Debugging with Google Search

As a SAS programmer, I've used an assortment of debugging techniques with varying degrees of success. A debugging technique that I've used more often than not is to use the power of Google® search. By copying an error message from the SAS Log and pasting it into the Google search box, the likelihood of finding a satisfactory solution to many, if not most, of the errors produced by errant SAS programs increases significantly. The following example illustrates using Google Search to find results related to "sas error message variable is uninitialized."



sas error message variable is uninitialized

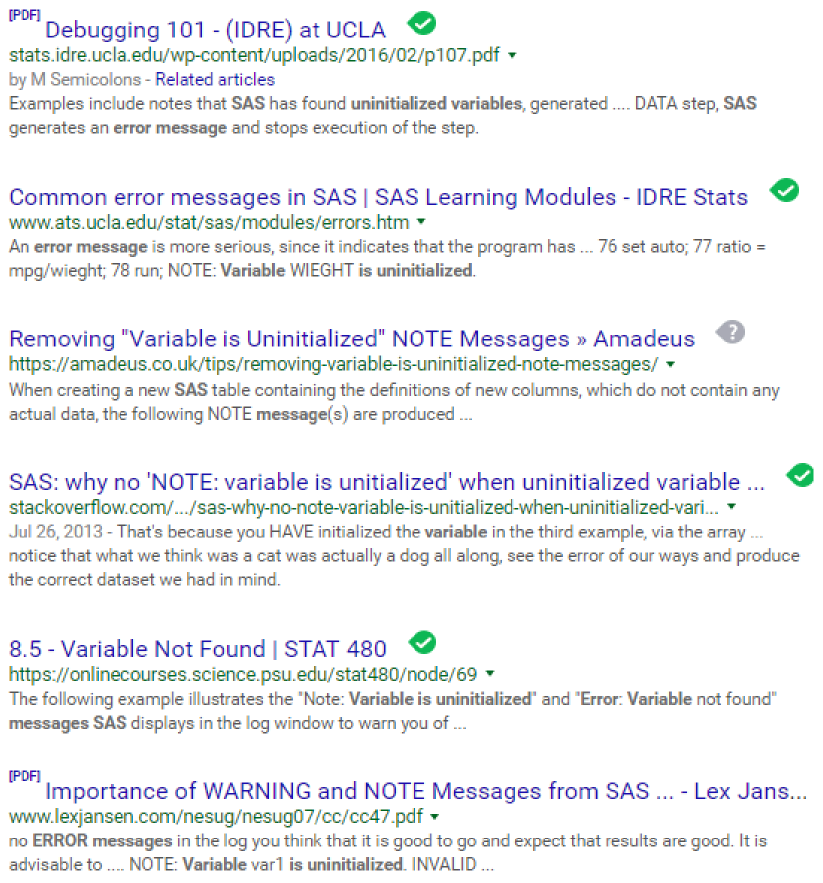
All Videos News Images Shopping More Settings Tools

About 12,200 results (0.66 seconds)

[PDF] Errors, Warnings, and Notes (Oh My): A Practical Guide to ... - SAS
www2.sas.com/proceedings/sugi22/BEGTUTOR/PAPER68.PDF
by SJ Slaughter - Cited by 1 - Related articles
NOTE: Variable NASPEC is uninitialized. When SAS is unable to find a variable in a DATA step, SAS prints the variable-is-uninitialized message. Then SAS creates the variable, sets its values to missing for all observations, and runs the DATA step.

[PDF] 232-30: Explaining Unexpected Log Messages and Output ... - SAS
www2.sas.com/proceedings/sugi30/232-30.pdf
the ones like "The variable XYZ is uninitialized" or "The variable ABC in the ... SAS programs generate numerous notes, warnings, and error messages in the ...

SAS tips & tricks #9 – NOTE: Variable X is uninitialized. - Statskom
statskom.com/sas-tips-tricks-9-note-variable-x-is-uninitialized/
Oct 9, 2014 - The "NOTE: Variable X is uninitialized" message is a result of trying to run a DATA Step which utilizes a variable that is not present in the input ...


[PDF] Debugging 101 - (IDRE) at UCLA
stats.idre.ucla.edu/wp-content/uploads/2016/02/p107.pdf
by M Semicolons - Related articles
Examples include notes that **SAS** has found **uninitialized variables**, generated .... DATA step, **SAS** generates an **error message** and stops execution of the step.

Common error messages in SAS | SAS Learning Modules - IDRE Stats
www.ats.ucla.edu/stat/sas/modules/errors.htm
An **error message** is more serious, since it indicates that the program has ... 76 set auto; 77 ratio = mpg/wieght; 78 run; NOTE: **Variable** WIEGHT **is uninitialized**.

Removing "Variable is Uninitialized" NOTE Messages » Amadeus
https://amadeus.co.uk/tips/removing-variable-is-uninitialized-note-messages/
When creating a new **SAS** table containing the definitions of new columns, which do not contain any actual data, the following NOTE **message**(s) are produced ...

SAS: why no 'NOTE: variable is unitialized' when uninitialized variable ...
stackoverflow.com/.../sas-why-no-note-variable-is-unitialized-when-uninitialized-vari...
Jul 26, 2013 - That's because you HAVE initialized the **variable** in the third example, via the array ... notice that what we think was a cat was actually a dog all along, see the error of our ways and produce the correct dataset we had in mind.

8.5 - Variable Not Found | STAT 480
https://onlinecourses.science.psu.edu/stat480/node/69
The following example illustrates the "Note: **Variable is uninitialized**" and "**Error**: **Variable** not found" **messages SAS** displays in the log window to warn you of ...

[PDF] Importance of WARNING and NOTE Messages from SAS ... - Lex Jans...
www.lexjansen.com/nesug/nesug07/cc/cc47.pdf
no **ERROR messages** in the log you think that it is good to go and expect that results are good. It is advisable to .... NOTE: **Variable** var1 **is uninitialized**. INVALID ...

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 of using Google Search to understand, fix, and resolve errors and warnings. The expectations are to improve the likelihood of preventing, fixing and/or resolving program errors allowing code to work as intended.

References

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 (2016); *"SAS® Debugging 101,"* Proceedings of the 2016 SAS Global Forum (SGF) Conference; Software Intelligence Corporation, Spring Valley, California, USA.

Lafler, Kirk Paul (2015); *"SAS® Debugging 101,"* Proceedings of the 2015 SAS Global Forum (SGF) Conference; Software Intelligence Corporation, Spring Valley, California, 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.

Wicklin, Rick (2010); "*How to Find and Fix Programming Errors*", http://blogs.sas.com/content/iml/2010/12/13/how-to-find-and-fix-programming-errors.html .

Acknowledgments

The author thanks William E. Benjamin Jr. and Frank Canale, Techniques & Tutorials Section Co-Chairs, for accepting my abstract and paper; Rob Howard, PharmaSUG 2017 Academic Chair and Ellen Brookstein, PharmaSUG 2017 Operations Chair; the PharmaSUG Executive Board, and the PharmaSUG 2017 Conference Committee for organizing and supporting a great conference!

Trademark Citations

About the Author

Kirk Paul Lafler is an entrepreneur, consultant and founder of Software Intelligence Corporation, and has been using SAS since 1979. Kirk is a SAS Certified Professional, provider of IT consulting services, professor at UC San Diego Extension and educator to SAS users around the world, mentor, and emeritus sasCommunity.org Advisory Board member. As the author of six books including Google® Search Complete (Odyssey Press. 2014) and PROC SQL: Beyond the Basics Using SAS, Second Edition (SAS Press. 2013); Kirk has written hundreds of papers and articles; been an Invited speaker and trainer at hundreds of SAS International, regional, special-interest, local, and in-house user group conferences and meetings; and is the recipient of 25 "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 Analyst, Educator and Author
E-mail: KirkLafler@cs.com
LinkedIn: http://www.linkedin.com/in/KirkPaulLafler
Twitter: @sasNerd