Abstract
As a SAS® user, you’ve probably experienced first-hand more than your share of program code bugs, and realize that debugging SAS program errors and warnings can, at times, be a daunting task. This paper and presentation explores the world of SAS errors and warnings, provides important information about syntax errors, input and output data sources, system-related default specifications, and logic scenarios specified in program code. Attendees will learn how to apply effective techniques to understand, fix, and resolve errors and warnings, allowing 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, we expect to improve our 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.

<table>
<thead>
<tr>
<th>Type of Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax Error</td>
<td>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.</td>
</tr>
<tr>
<td>Semantic Error</td>
<td>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.</td>
</tr>
<tr>
<td>Execution-time Error</td>
<td>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.</td>
</tr>
<tr>
<td>Data Error</td>
<td>An error that is generated when one or more data values do not match an INPUT statement specification.</td>
</tr>
<tr>
<td>Macro-Related Error</td>
<td>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.</td>
</tr>
<tr>
<td>Logic Error</td>
<td>An error that does not have a Note, Warning or Error associated with it, but contains erroneous or unexpected results. Examples include using a “&lt;” when a “&gt;” comparison operator should have been specified.</td>
</tr>
</tbody>
</table>

Table 1. Error Classification and Description
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.

<table>
<thead>
<tr>
<th>Errors During Compilation Phase</th>
<th>Errors During Execution Phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax Error</td>
<td>Execution-time Error</td>
</tr>
<tr>
<td>Semantic Error</td>
<td>Data Error</td>
</tr>
<tr>
<td>Macro-Related Error</td>
<td>Macro-Related Error</td>
</tr>
</tbody>
</table>

Table 2. Types of Errors and Error Processing Phases

Errors that SAS Can't Help With
Users, with the aid of the SAS Log, are able to determine the cause of many types of errors, but there is one particular type of error that SAS is able to offer very little help with – Logic errors. A good strategy to use in the detection and resolution of Logic 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 logic 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).

Examining 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:**
```sas
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:**
```sas
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:**
```sas
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:**
```sas
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:**
```sas
options MSGLEVEL=I;
data libref.movies;
set work.movies;
< other SAS statements >;
run;
```
**Using NOTES / NONOTES System Option**
The NOTES system option can be used to tell SAS to print all notes to the SAS Log.

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

**Using REPLACE / NOREPLACE System Option**
The NOREPLACE system option can be used to prevent the replacement of permanent data sets.

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

**Using SOURCE2 / NOSOURCE2 System Option**
The SOURCE2 system option can be used to write all included source code (used with the %include statement) to the SAS Log.

**Example:**
```sas
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;
%MIMO 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

```sas
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):

<table>
<thead>
<tr>
<th></th>
<th>Real Time</th>
<th>CPU Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>real time</td>
<td>0.00</td>
<td>0.01</td>
</tr>
</tbody>
</table>

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, and system options to debug macro code. 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, Austin, Texas, USA.


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


Acknowledgments
I want to thank Isabel Jacobson, SAS Essentials Section Chair, for accepting my abstract and paper; Cynthia LeardMann, Academic Chair; MaryAnne Depesquo, Operations Chair; and the WUSS 2013 Conference Committee for organizing a great conference!

Trademark Citations
SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Benchworkzz® for SAS® product or service names are registered trademarks of Benchworkzz, LLC, Austin, Texas USA. Other brand and product names are trademarks of their respective companies.

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 five books including PROC SQL: Beyond the Basics Using SAS, Second Edition (SAS Press 2013) and PROC SQL: Beyond the Basics Using SAS (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 22 “Best” contributed paper, hands-on workshop (HOW), and poster awards.

Comments and suggestions can be sent to:

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