PharmaSUG 2025 Paper AP-161

Get Tipsy with Debugging Tips for SAS® Code: The After Party

Lisa Mendez, Army MEDCOM; Richann Jean Watson, DataRich Consulting

ABSTRACT

You've spent hours crafting your SAS® code, or perhaps you've inherited a jumbled mess from someone else, and when you run it, it "explodes" like an ill-fated cocktail experiment and leaves a bad taste in your mouth! Mastering the art of debugging not only makes coding a smoother experience but also spares you from needing a stiff drink to cope with the frustration.

This paper will discuss tips, tricks, and techniques for debugging your SAS code. It will show you how to use the debugger in both SAS Display Manager and SAS Enterprise Guide, akin to a bartender knowing just the right tools to mix the perfect drink. It will also describe how to use PUTLOG and PUT statements, along with their key differences—like choosing between a whiskey neat and a whiskey sour.

Additional debugging aids such as using OPTIONS, and macro variables will be discussed, ensuring you have a well-stocked bar of techniques at your disposal. As a bonus, the paper will also provide some common error messages and their meanings—your personal cocktail menu of debugging solutions.

INTRODUCTION

Debugging SAS code can often feel like navigating a crowded bar, especially when you've spent countless hours crafting your scripts or inheriting a chaotic tangle from someone else. When your code "explodes" like a disastrous cocktail experiment, it leaves a bitter taste and a heap of frustration. Mastering the art of debugging not only smooths out the coding process but also saves you from needing a stiff drink to cope with the stress.

This paper aims to equip you with essential tips, tricks, and techniques for debugging your SAS code. We'll explore how to use the debugger in both SAS Display Manager and SAS Enterprise Guide. Additionally, we'll delve into the use of PUTLOG and PUT statements, highlighting their key differences.

Furthermore, we'll discuss additional debugging aids such as using OPTIONS (such as MPRINT, MLOGIC, and SYMBOLGEN) ensuring you have a well-stocked bar of techniques at your disposal. As a bonus, we'll provide a guide to common error messages and their meanings. With these tools, you'll be well-prepared to tackle any coding challenge with confidence and precision.

THE DEBUGGER

Within SAS Display Manager and SAS Enterprise Guide, there's a powerful tool known as the debugger. Think of it as the expert bartender overseeing the creation of a finely crafted cocktail—your DATA step. The debugger allows you to interactively examine how each ingredient (or line of code) is mixed, interpreted, and refined, ensuring the perfect balance for every row of data.

Just like a drink recipe must come together correctly before being served, your SAS code must successfully compile. Once it does, the debugger springs into action, letting you follow along, step by step, as the flavors—or data elements—come together to create the final masterpiece.

The DATA step debugger in SAS Display Manager can be accessed by typing '`/ debug`' on the DATA step statement as seen in SAS Program 1.

```
data class;
   set SASHELP.CLASS;
   where SEX = 'F' and AGE <= 13;
run;
```

SAS Program 1: SAS Display Manager Editor Window

Upon execution of the code, the debugger tool is opened (Image 1). You can then provide commands on the prompt, which is at the bottom of the DEBUGGER LOG, highlighted in pink in Image 1.

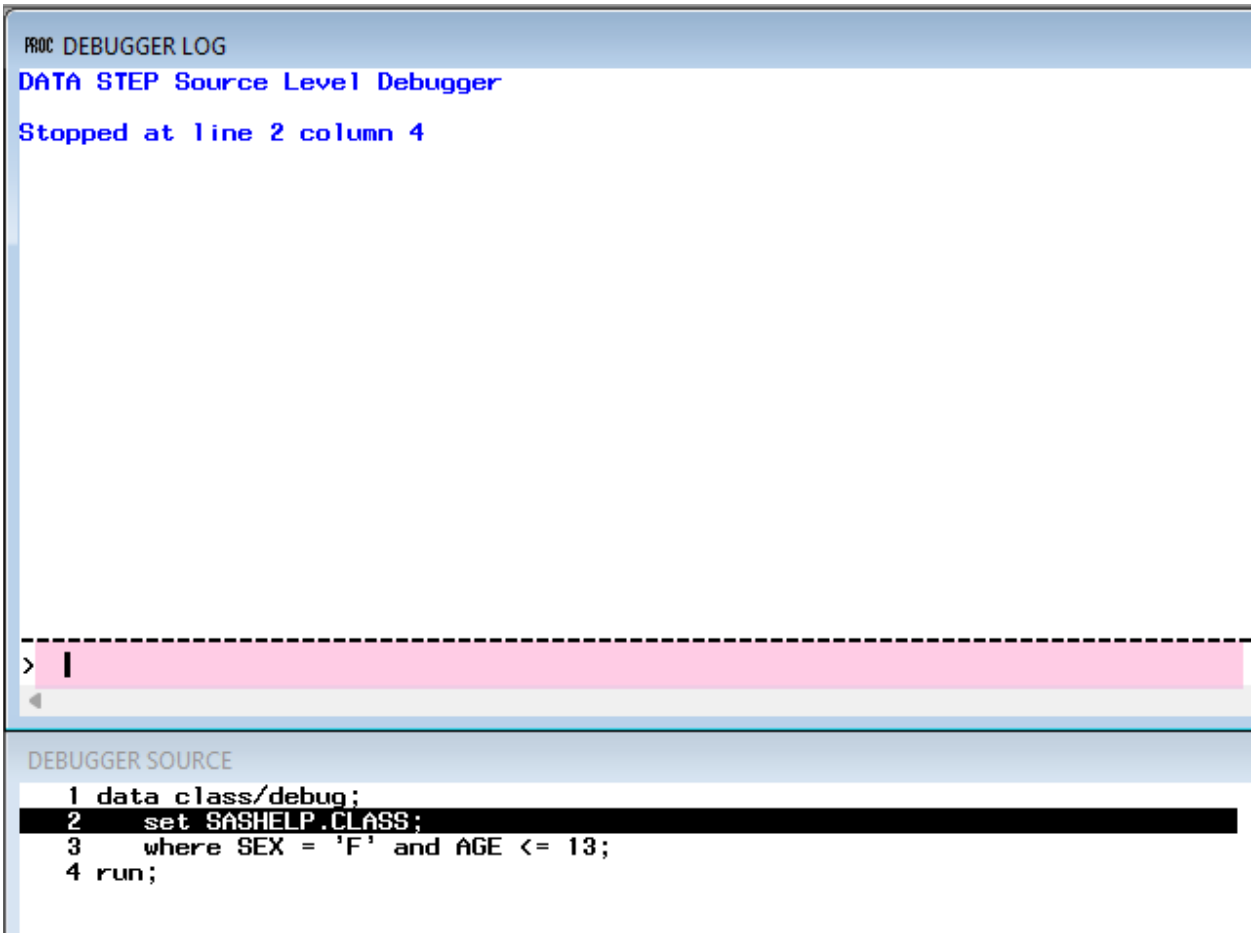

Image 1: SAS Display Manager DATA Step Debugger

Commands that can be used include:

- STEP: This allows you to step through the SAS code.
- EXAMINE <VARIABLE>: This displays the values of the variable(s) indicated. To see values for all the variables, you can use _ALL_. You can also specify a format for a specific variable if necessary.
- QUIT: This terminates the debugger session

Image 2 shows a few commands that were executed. The STEP command stepped to the next line of code and the EXAMINE command allowed for the examination of the AGE variable in the first record. Since another STEP command was not given, the EXAMINE command for AGE with a format applied still corresponded to the first record. The last EXAMINE command indicated that the values for all the variables are to be displayed for the first record. The process of STEP and EXAMINE can be repeated as many times as necessary to what is being entered into the PDV and what is eventually being written to the data set.

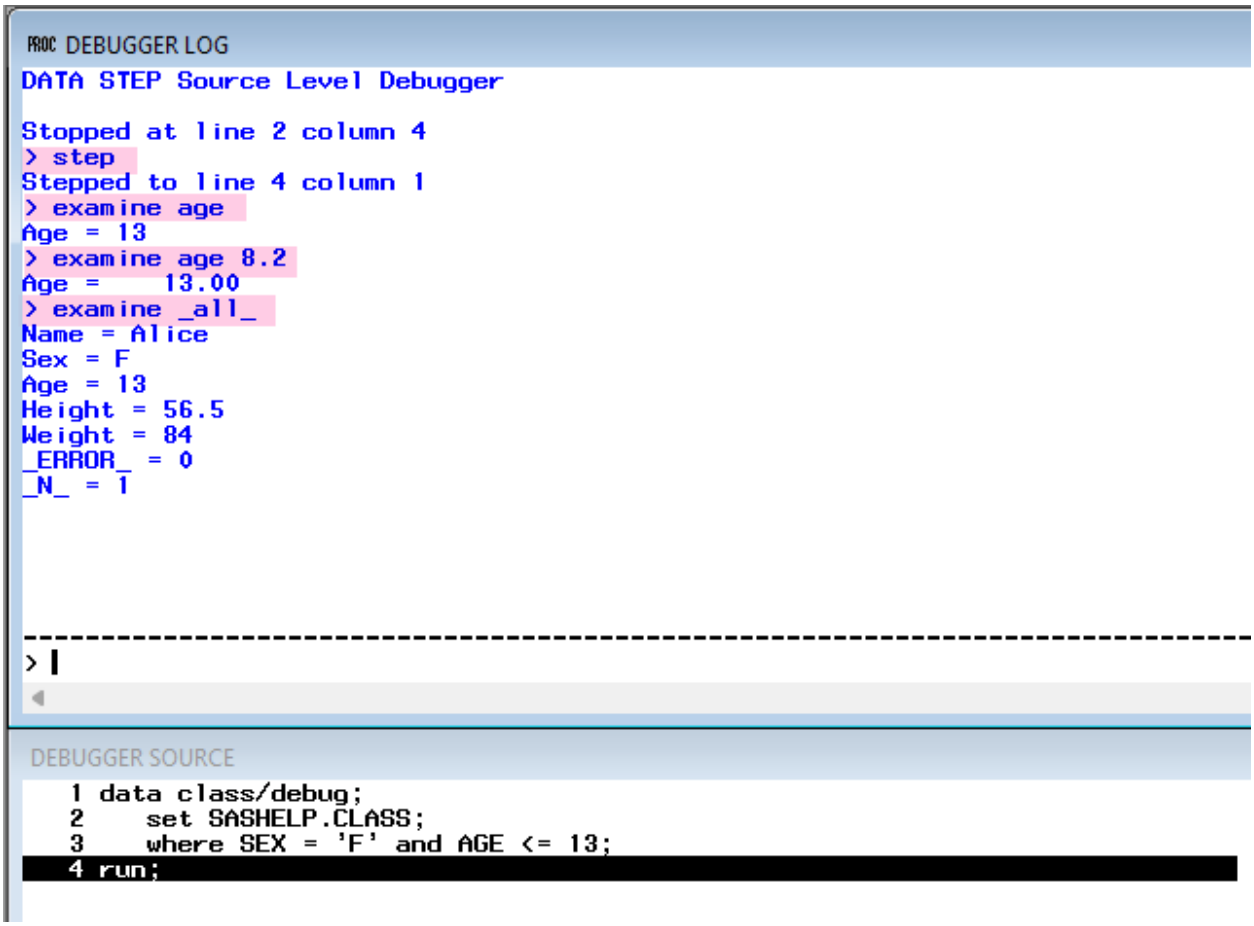

Image 2: Illustration of SAS Display Manager DATA Step Debugger

To access the DATA step debugger in SAS Enterprise Guide, you need to click on the 'Debug' tool in the menu bar. This gives you a green bar with a bug icon in the margin for the portions of code (i.e., DATA steps) that can be debugged (Image 3).

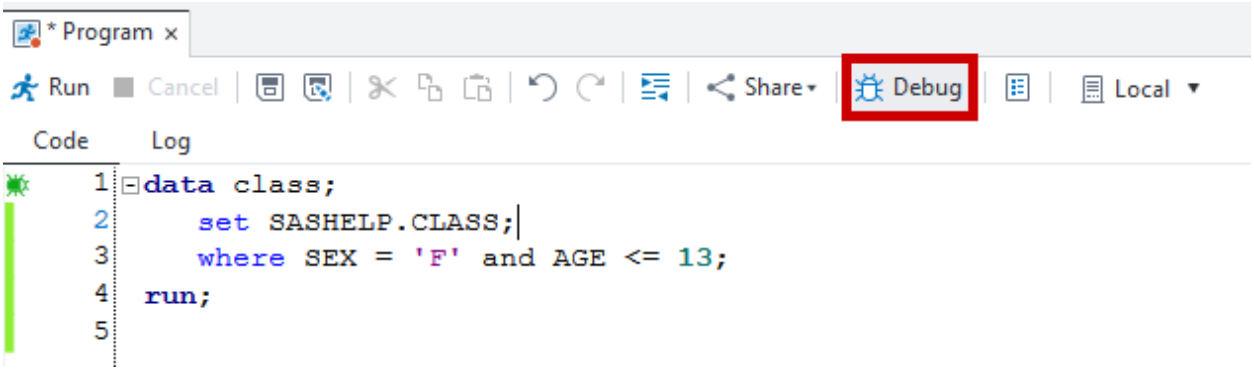

Image 3: SAS Enterprise Guide Program Window

Once the debugger tool is activated, you can click on the green bar next to the code that you want to step through. This will open the DATA Step Debugger (Image 4).

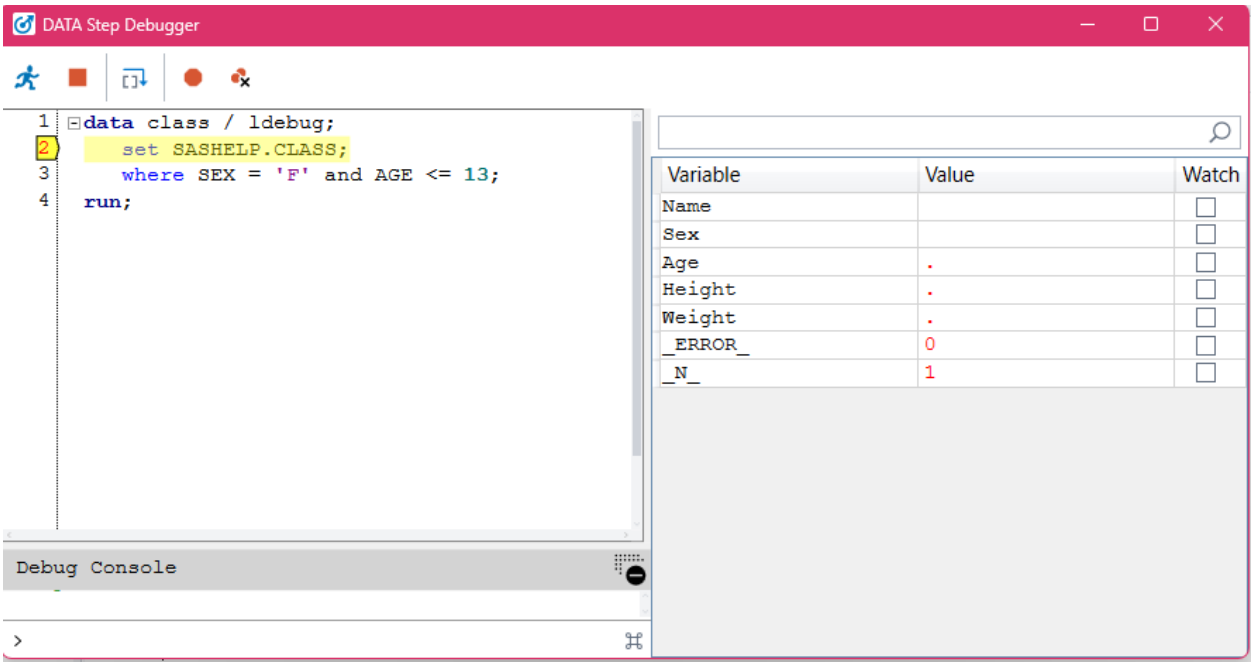

Image 4: SAS Enterprise Guide DATA Step Debugger Window

Within the DATA Step Debugger tool, you can step through the code line by line by clicking on so that you can examine the program data vector (PDV) to see how SAS is interpreting the code. Refer to Image 5 for an illustration of the debugger in SAS Enterprise Guide.

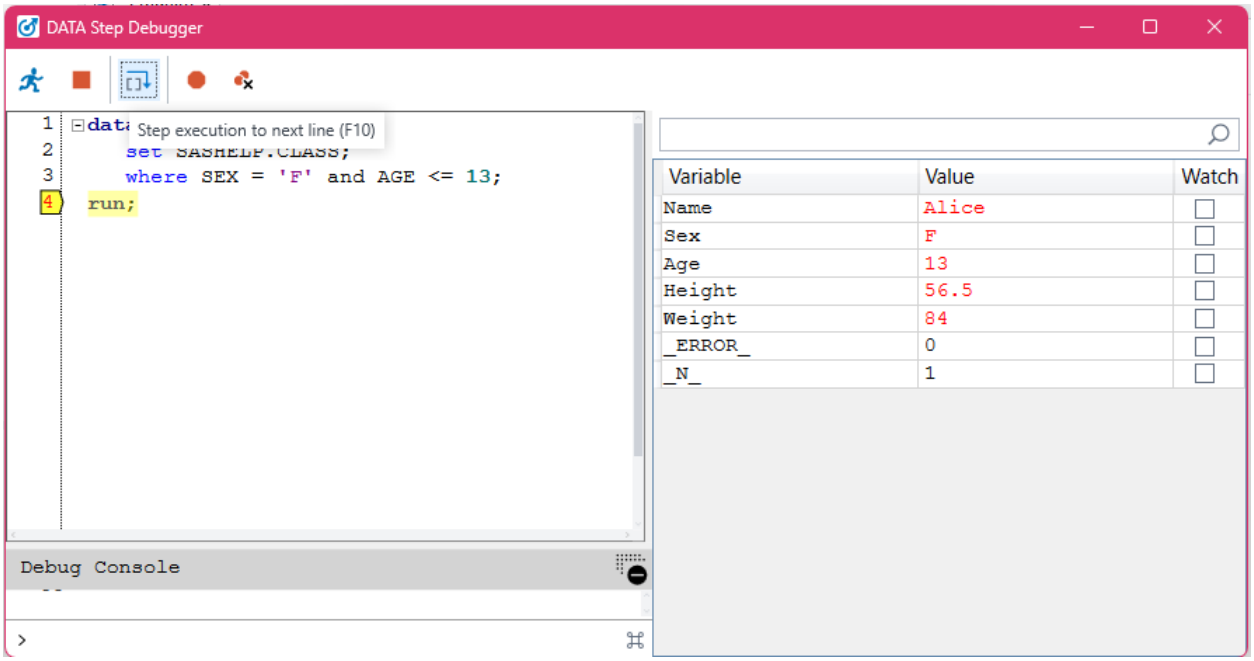

Image 5: Illustration of SAS Enterprise Guide DATA Step Debugger

While there's much more you can do with the debugger tool—like setting breaks, watching variables, and calculating expressions—getting a solid grasp of the basics is like mastering a signature cocktail. Once you know the key ingredients, refining the mix becomes much easier. For a deep dive into the DATA Step Debugger, check out Chris Hemedinger's blog, *Using the DATA step debugger in SAS Enterprise Guide* (Hemedinger, 2016).

Keep in mind, the SAS debugger is exclusive to SAS Display Manager and SAS Enterprise Guide—think of it as a specialty drink only served at select bars. Unfortunately, SAS Studio doesn't have a built-in debugger, so you'll need to rely on other classic techniques, such as using PUT or PUTLOG, to troubleshoot and fine-tune your recipe.

PUTLOG AND PUT STATEMENTS

In SAS programming, PUT and PUTLOG statements are your trusty sidekicks, ready to jump in and help debug your DATA step code. These two tools have their own unique personalities, but both are indispensable when you're unraveling the mysteries of your program. The main distinction? The PUT statement is like the extrovert at the party—it loves the spotlight and will shout its message to any open destination. PUTLOG, on the other hand, is the focused introvert, delivering messages exclusively to the log where the real action happens. Let's dive into their quirks and abilities!

PUT Statements: The Multitasker

The PUT statement is flexible and can:

- Print variable values at different stages of your code, giving you a play-by-play of what's happening.
- Deliver custom messages to track program flow and flag important milestones.
- Handle not just macros but also DATA step operations, proving its versatility.

For instance, imagine you need to track the value of a variable during execution. You could use PUT to output a message like:

```
data example;
    x = 5;
    y = x * 10;
    put "The value of y is: " y;
run;
```

SAS Program 2: PUT Statement

```
1    data example;
2        x = 5;
3        y = x * 10;
4        put "The value of y is: " y;
The value of y is: 50
5    run;

NOTE: The data set WORK.EXAMPLE has 1 observations and 2 variables.
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

Log of SAS Program 2: Put Statement

The PUT statement prints "The value of y is: 50" to the destination of your choice, typically the log but you can write to an external file (as seen in our next example), making it easy to keep tabs on your variables.

PUTLOG: The Laser-Focused Debugger

PUTLOG takes a slightly different approach. Its mission is singular: to communicate directly with the SAS log. This makes it a fantastic tool for tracking the flow of your DATA step program and inspecting variable values with precision. Think of it as your personal mixologist, trained to create that special cocktail, specifically made to your taste.

PUTLOG is particularly useful within conditional logic. You can log messages only when specific

conditions are met or when the data is unusual (Wicklin, 2023), helping you zero in on issues. For instance:

```
data data dates;
  infile datalines delimiter = ',';
  length SRTDT $20 ENDDT $20;
  input SRTDT ENDDT;

  if first(SRTDT) = '-' then putlog 'ERROR: invalid start year' _n_= SRTDT=;
  else if 4 < lengthn(SRTDT) < 10 then
       putlog 'WARNING: partial start date - missing day' _n_= SRTDT=;

  if first(ENDDT) = '-' then put 'ERROR: invalid end year' _n_= ENDDT=;
  else if 4 < lengthn(ENDDT) < 10 then put 'WARNING: partial end date - missing day'
_n_= ENDDT=;

  file 'C:\Users\gonza\OneDrive -
datarichconsulting.com\Desktop\Conferences\Drafts\The After Party\dates.txt';
  if lengthn(SRTDT) = 10 then putlog 'NOTE: Complete start date' _n_= SRTDT=;
  if lengthn(ENDDT) = 10 then put 'NOTE: Complete end date' _n_= ENDDT=;

datalines;
2021, 2021
2021-02, 2021-03
2021-07-09, 2022-09
2021-07, 2022-08-09
-, -
;
run;
```

SAS Program 3: PUTLOG Statement

```
...

NOTE: The file 'C:\Users\gonza\OneDrive -
datarichconsulting.com\Desktop\Conferences\Drafts\The After Party\dates.txt' is:
      Filename=C:\Users\gonza\OneDrive -
datarichconsulting.com\Desktop\Conferences\Drafts\The After Party\dates.txt,
      RECFM=V,LRECL=32767,File Size (bytes)=0,
      Last Modified=01Apr2025:22:57:18,
      Create Time=01Apr2025:22:52:53

WARNING: partial start date - missing day_N_=2 SRTDT=2021-02
WARNING: partial end date - missing day_N_=2 ENDDT=2021-03
WARNING: partial end date - missing day_N_=3 ENDDT=2022-09
NOTE: Complete start date_N_=3 SRTDT=2021-07-09
WARNING: partial start date - missing day_N_=4 SRTDT=2021-07
ERROR: invalid start year_N_=5 SRTDT=-
ERROR: invalid end year_N_=5 ENDDT=-
NOTE: 1 record was written to the file 'C:\Users\gonza\OneDrive -
datarichconsulting.com\Desktop\Conferences\Drafts\The After
      Party\dates.txt'.
      The minimum record length was 45.
      The maximum record length was 45.
NOTE: The data set WORK.DATES has 5 observations and 2 variables.
NOTE: DATA statement used (Total process time):
      real time           0.04 seconds
      cpu time            0.03 seconds
```

Log of SAS Program 3: PUTLOG Statement with Severity Level

In this case, the PUTLOG statement writes multiple statements to the log: the statement highlighted in green is the NOTE, and the blue highlighted statements are the ERROR messages.

The yellow highlighted section of the log shows that the PUT statement from the code was written to an external text file.

PUTLOG also supports severity levels. You can categorize your log messages as NOTE, WARNING, or ERROR, adding an extra layer of clarity when things go awry. With %PUTLOG, SAS knows to treat **ERROR** messages as actual errors, potentially halting execution or prompting further debugging. Rick Wicklin's blog (Wicklin, 2023) provides an additional example of how to use the PUTLOG statement, making it a valuable resource for further insights.

OPTIONS

MLOGIC

The MLOGIC option in SAS is a powerful debugging tool that provides detailed insights into the execution of macro logic. When enabled, it writes information about the execution of macro statements to the SAS log, allowing you to trace the flow of macro processing step-by-step. To use the MLOGIC option, you simply include the statement OPTIONS MLOGIC; at the beginning of your SAS program or before the macro execution you wish to debug. It only needs to be executed once and will stay active until it is turned off. You don't need to run the Option MLOGIC prior to every macro call. In addition, note that it must be invoked prior to the macro call, and not necessarily prior to the macro definition. Once activated, the SAS log displays detailed messages that reveal the macro name being invoked, the values of parameters passed to the macro, and the internal flow of execution, including iterative or conditional steps. This can help identify errors or unexpected behavior within macros, such as incorrect parameter values or logic that does not execute as intended. By providing clear and specific details about macro processing, the MLOGIC option enables you to efficiently diagnose and resolve issues in your SAS programs, making it an invaluable tool for debugging complex macro-based workflows.

This example demonstrates the functionality of the MLOGIC option.

```
options mlogic;
%macro calc_per(principal = , rate = , years =);
   %if &rate =   %then %do;
      %let interest = %sysevalf(&principal * 0.0799 * &years);
   %end;
   %else %do;
      %let interest = %sysevalf(&principal * &rate * &years);
   %end;

   %put The calculated interest is &interest;
%mend calc_per;
options mlogic;
%calc_per(principal = 1000, rate = 0.05, years = 3)
%calc_per(principal = 1000, years = 3)
```

SAS Program 4: Options MLOGIC

The log will display detailed information about the macro's execution, including the values of the parameters and the steps performed.

```
...

MLOGIC(CALC_PER):  Beginning execution.
MLOGIC(CALC_PER):  Parameter PRINCIPAL has value 1000
MLOGIC(CALC_PER):  Parameter RATE has value 0.05
MLOGIC(CALC_PER):  Parameter YEARS has value 3
MLOGIC(CALC_PER):  %IF condition &rate = is FALSE
MLOGIC(CALC_PER):  %LET (variable name is INTEREST)
MLOGIC(CALC_PER):  %PUT The calculated interest is &interest
```

```
The calculated interest is 150
MLOGIC(CALC_PER):  Ending execution.
14          %calc_per(principal = 1000, years = 3)
MLOGIC(CALC_PER):  Beginning execution.
MLOGIC(CALC_PER):  Parameter PRINCIPAL has value 1000
MLOGIC(CALC_PER):  Parameter YEARS has value 3
MLOGIC(CALC_PER):  Parameter RATE has value
MLOGIC(CALC_PER):  %IF condition &rate = is TRUE
MLOGIC(CALC_PER):  %LET (variable name is INTEREST)
MLOGIC(CALC_PER):  %PUT The calculated interest is &interest
The calculated interest is 239.7
MLOGIC(CALC_PER):  Ending execution.
```

Log of SAS Program 4: Options MLOGIC

The MLOGIC option breaks down each step of the macro execution, making it easy to trace how the parameters were passed and how the calculations were performed. This example demonstrates how MLOGIC can provide clarity and help debug macro-related issues.

MPRINT

Another great option to use when running macros in SAS is the MPRINT option. This option displays the generated SAS code produced by macro execution in the SAS log. When using this option in your code, you can see exactly how macro variables and statements are resolved into standard SAS code. This visibility is helpful for troubleshooting and understanding the logic of macros, as it allows you to verify that the generated code aligns with your expectations.

This example demonstrates the functionality of the MPRINT option.

```
options mprint;
%macro calc_per(principal = , rate = );
   %let __rate = %sysfunc(translate(&rate, '_', '.'));
   data rate_&__rate;
      principal = &principal;
      rate = &rate;
      %do year = 5 %to 8;
         year = &year;
         interest = &principal * &rate * &year;
         output;
      %end;
   run;
%mend calc_per;
options mlogic;
%calc_per(principal = 1000, rate = 0.05)
```

SAS Program 5: Options MPRINT

The log output shows the resolved SAS code that was generated from the macro.

```
...

MLOGIC(CALC_PER):  Beginning execution.
MLOGIC(CALC_PER):  Parameter PRINCIPAL has value 1000
MLOGIC(CALC_PER):  Parameter RATE has value 0.05
MLOGIC(CALC_PER):  %LET (variable name is __RATE)
MPRINT(CALC_PER):   data rate_0_05;
MPRINT(CALC_PER):   principal = 1000;
MPRINT(CALC_PER):   rate = 0.05;
MLOGIC(CALC_PER):  %DO loop beginning; index variable YEAR; start value is 5; stop
value is 8; by value is 1.
MPRINT(CALC_PER):   year = 5;
MPRINT(CALC_PER):   interest = 1000 * 0.05 * 5;
MPRINT(CALC_PER):   output;
```

```
MLOGIC(CALC_PER):  %DO loop index variable YEAR is now 6; loop will iterate again.
MPRINT(CALC_PER):   year = 6;
MPRINT(CALC_PER):   interest = 1000 * 0.05 * 6;
MPRINT(CALC_PER):   output;
MLOGIC(CALC_PER):  %DO loop index variable YEAR is now 7; loop will iterate again.
MPRINT(CALC_PER):   year = 7;
MPRINT(CALC_PER):   interest = 1000 * 0.05 * 7;
MPRINT(CALC_PER):   output;
MLOGIC(CALC_PER):  %DO loop index variable YEAR is now 8; loop will iterate again.
MPRINT(CALC_PER):   year = 8;
MPRINT(CALC_PER):   interest = 1000 * 0.05 * 8;
MPRINT(CALC_PER):   output;
MLOGIC(CALC_PER):  %DO loop index variable YEAR is now 9; loop will not iterate
again.
MPRINT(CALC_PER):   run;

NOTE: The data set WORK.RATE_0_05 has 4 observations and 4 variables.
NOTE: DATA statement used (Total process time):
      real time           0.02 seconds
      cpu time            0.01 seconds
```

Log of SAS Program 5: Options MPRINT

	principal	rate	year	interest
1	1000	0.05	5	250
2	1000	0.05	6	300
3	1000	0.05	7	350
4	1000	0.05	8	400

Output Dataset of SAS Program 5

The MPRINT option makes it easy to verify the code being executed, helping you debug macros and ensure the logic works as intended.

SYMBOLGEN

One last option that is useful when debugging macro code is the SYMBOLGEN option. When enabled this option writes detailed information to the SAS log about how macro variables are resolved. Specifically, it displays the values of macro variables as they are substituted in your code, allowing you to trace and verify their behavior during execution. This option is especially helpful for identifying and resolving issues when macro variables produce unexpected or incorrect values.

This example demonstrates the functionality of the SYMBOLGEN option.

```
options symbolgen;
%macro calc_per(principal = , rate = , years =);
   %if &rate =   %then %do;
      %let interest = %sysevalf(&principal * 0.0799 * &years);
   %end;
   %else %do;
      %let interest = %sysevalf(&principal * &rate * &years);
   %end;

   %put The calculated interest is &interest;
%mend calc_per;
%calc_per(principal = 1000, rate = 0.05, years = 3)
%calc_per(principal = 1000, years = 3)
```

SAS Program 6: Options SYMBOLGEN

Sample of the Log output. The log output shows how SYMBOLGEN traces the resolution of macro variables, displaying their values as they are substituted into the code.

```
...

SYMBOLGEN:  Macro variable RATE resolves to 0.05
SYMBOLGEN:  Macro variable PRINCIPAL resolves to 1000
SYMBOLGEN:  Macro variable RATE resolves to 0.05
SYMBOLGEN:  Macro variable YEARS resolves to 3
SYMBOLGEN:  Macro variable INTEREST resolves to 150
The calculated interest is 150
13         %calc_per(principal = 1000, years = 3)
SYMBOLGEN:  Macro variable RATE resolves to
SYMBOLGEN:  Macro variable PRINCIPAL resolves to 1000
SYMBOLGEN:  Macro variable YEARS resolves to 3
SYMBOLGEN:  Macro variable INTEREST resolves to 239.7
The calculated interest is 239.7
```

Log of SAS Program 6: Options SYMBOLGEN

This insight makes it easier to verify the correctness of the variables and debug any issues related to their usage. Although this sample seems straightforward, it is very useful when using macro variables that are assigned from a data set or system variables.

COMMON ERROR MESSAGES

Below are some common error messages that we have encountered during our career (and yes, we still encounter them).

MISSING SEMICOLON

The dreaded missing semicolon! It can be any SAS programmer's nemesis. It is one of the most common and frustrating errors encountered by programmers. A single missing semicolon can cause the program to misinterpret the code, leading to a cascade of errors. For example, if a semicolon is missing after a statement, SAS may merge that statement with the subsequent code, generating confusing syntax errors that can be difficult to debug. The error messages associated with a missing semicolon often indicate a "syntax error" and list unexpected or missing tokens (e.g.; (, *, +). These messages usually highlight the line where SAS thinks the problem exists, but because SAS processes code sequentially, the actual issue may occur earlier in the program. The missing semicolon can also lead to incomplete data sets, warnings, or the failure of subsequent steps.

If you ever encounter the following error message, note, and warning, it is most likely due to a missing semicolon in your code.

```
ERROR: Syntax error, expecting one of the following: ;, (, /, ||, :, _,
*, **, +, -, ....
NOTE: The SAS System stopped processing this step because of errors.
WARNING: The data set WORK.EXAMPLE may be incomplete. When this step
was stopped there were 0 observations and 0 variables.
NOTE: The PROCEDURE PRINT used 0 pages.
```

What's the golden rule for dodging this pesky error? Give your code a good once-over before you hit submit. And hey, consider a missing semicolon as your initiation rite—because no SAS programmer is truly seasoned until they've stared down this classic mistake!

UNRECOGNIZED STATEMENT

The "Unrecognized Statement" error in SAS is basically SAS's way of saying, "I have no idea what you're trying to tell me." It usually pops up when your code has a typo, a misspelled keyword, or just isn't quite formatted to SAS's liking. For instance, if you mistakenly type "dtata" instead of "data", SAS will look at you like, "What in the world is that supposed to mean?"

Sometimes this error also arises if your code is in the wrong place, or if SAS is left hanging because you forgot an essential piece, like a semicolon or a matching RUN statement. When this happens, DON'T WORRY! Just comb through the log and review the line SAS flagged (and maybe a few lines above it) for typos, missing keywords, or misplaced statements. Think of it as an initiation rite into the "Don't drink and code" club (or better yet, "Drink and debug").

Here is a very easy example to show you how this error message shows up in the log.

```
dtata example;
    x = 5;
    y = 10;
run;
```

SAS Program 7: Unrecognized Statement

```
1    dtata example;
      -----
      22
ERROR 22-322: Syntax error, expecting one of the following: a valid SAS name.
ERROR 76-322: Syntax error, statement will be ignored.
2       x = 5;
3       y = 10;
4    run;

NOTE: The SAS System stopped processing this step because of errors.
```

Log of SAS Program 7: Unrecognized Statement

VARIABLE NOT FOUND

The "VARIABLE NOT FOUND" error in SAS occurs when the system tries to reference a variable that does not exist in the current data set or program context. This can happen for several reasons, including:

- **Misspelled Variable Name:** If you accidentally misspell the name of a variable, SAS won't recognize it and will throw this error (like someone throwing a drink in your face).
- **Variable Not Created:** If the variable hasn't been properly initialized or created in the data step or procedure, it won't be accessible.
- **Incorrect Data set Reference:** If you are working with multiple data sets and reference a variable from the wrong data set or one that doesn't contain the variable, the error will occur.

- **Logical Errors:** If a condition or calculation involves a variable that hasn't been defined, SAS will flag it as missing.

Let's look at an example:

```
data example;
    set dataset;
    if age > 30 then newvar = "Older";
    if gender = "Male" then group = "Males";
run;
```

SAS Program 8: Variable Not Found

```
1    data example;
2        set dataset;
3        if age > 30 then newvar = "Older";
4        if gender = "Male" then group = "Males";
ERROR: Variable GENDER not found.
NOTE: The SAS System stopped processing this step because of errors.
WARNING: The data set WORK.EXAMPLE may be incomplete. When this step was stopped there
were 0 observations and 0 variables.
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

Log of SAS Program 8: Variable Not Found

If the data set being referenced does not include the variable gender, SAS will output a "VARIABLE NOT FOUND" error.

DATA STEP ERRORS

You have your code written and it looks beautiful. You are ready to execute. It compiles fine, but then something goes wrong. These are known as execution-time errors. SAS has an automatic variable, _ERROR_, that is initially set to 0 for each iteration in a DATA step and that is set to 1 when an error is detected (Burlew, 2001). Although they may be referred to as execution-time errors, they may not always show up as an ERROR in the log. However, they should still be addressed. Below are some common execution-time errors:

- When using a BY statement in a data set, the input data set must be sorted by the BY variables otherwise it will produce the following ERROR message.

  ```
  ERROR: BY variables are not properly sorted on data set SASHELP.CLASS.
  ```

- When a variable is assigned to a data type via a length statement or attribute statement, but the data is of a different data type, then you get an ERROR message indicating that the variable has been assigned to both data types. Therefore, it is important to know your data and the data types associated with the variables.

  ```
  ERROR: Variable AGE has been defined as both character and
  numeric.
  ```

- When a variable is already defined within the program data vector (PDV) as numeric, but it is treated like a character variable, then you can end up with a NOTE indicating invalid numeric data or you can end up with a WARNING and a NOTE indicating that the variable was already defined as numeric and the data is invalid. Both of these log messages sets the _ERROR_ to 1. If the variables are meant to be character, then the first reference to the variable in the DATA step should define it as character.

```
NOTE: Invalid numeric data, 'Y' , at line 30 column 10.
Name=Alfred Sex=M Age=. Height=69 Weight=112.5 _ERROR_=1 _N_=1

WARNING: Variable itemid1 has already been defined as numeric.
42          2357 LIB0003 LIB0032 LIB0043 LIB0900 LIB1309
itemid1=. itemid2=. itemid3=. itemid4=. itemid5=. empno=2357 i=6
itemno=. _ERROR_=1 _N_=1
NOTE: Invalid data for itemid1 in line 43 6-12.
```

- When executing a statement whether it is a conditional logic or a calculation, it is important to ensure that the variables used in the statement are of the correct data type, otherwise, you can end up with one of the following NOTEs. SAS makes an attempt to convert the value in order to execute the statement, but in cases where the conversion is not possible, this can lead to additional error messages. To avoid these types of messages, you should either use various SAS functions to perform the conversion and prevent these messages or write the code so that it reflects the correct data type.

```
NOTE: Character values have been converted to numeric values at the
places given by: (Line):(Column).
5:12
NOTE: Numeric values have been converted to character values at the
places given by: (Line):(Column).
6:12
```

- When performing operations on missing values, this can lead to unexpected results. The following message can be due to invalid data, illegal mathematical operation, a missing variable or variable that is truly missing. The use of various SAS functions to check for missing values or functions that allow for mathematical operations even if one or more values are missing could be implemented in order to avoid this message.

```
NOTE: Missing values were generated as a result of performing an
operation on missing values.
```

- When doing a mathematical calculation such as division, it is important to check that you are not doing a division by zero. Although, SAS indicates this division by zero as a NOTE in the log, it still sets the _ERROR_ to 1. There are two approaches to getting around this log message. You can use IF-THEN logic to check that the divisor is not zero, or you can use the DIVIDE function. With the DIVIDE function, it returns 'I' for infinity when there is a division by zero.

```
NOTE: Division by zero detected at line 4 column 10.
X=10 Y=0 Z=. _ERROR_=1 _N_=1
NOTE: Mathematical operations could not be performed at the following
places. The results of the operations have been set to missing values.
```

MISMATCHED QUOTATION MARKS OR COMMENT TAGS

You have probably seen this dreaded ERROR in your logs and have no idea what is tripping up the program and causing mayhem to your SAS session, that you want to shut down the SAS session and start a new one because you can't get anything to compile and/or execute properly.

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

After examining the code and/or log you realize we have unbalanced quotation marks or maybe an unmatched comment tag. If you have an unbalanced quotation mark, you can try submitting a quotation mark followed by "`; RUN;`". If it is unmatched comment tag (e.g., /* */), then you can try submitting the closing comment tag followed by "; RUN;".

If those still don't untangle the chaos, then before shutting down try typing the following sequence of characters.

```
/* '; * "; */;
quit;
```

OPEN CODE RECURSIVE

As you continue to build your SAS skills, you may start to incorporate macro language elements into your program. Even as experienced programmers, it can be easy to muck up your code by forgetting something as simple as a semicolon —like missing a key ingredient in a carefully crafted cocktail.. Well, that forgotten semicolon causes a different type of ERROR message when it involves the macro language facility. A forgotten semicolon can cause open code recursion. Open code recursion can occur when that little semicolon is a terminating semicolon, and it is left out and another macro language statement immediately follows. (Burlew, 2001). The following is an example of open code recursion. Notice the missing semicolon at the end of the first %LET.

```
%let order = _itt_resp
%let tst = %scan(&order, 2, '_');
```

This causes the macro processor to write the following message to the log

```
ERROR: Open code statement recursion detected.
```

Sometimes submitting a single semicolon will fix the problem, but there can be those cases where it doesn't and it is causing so much frustration that you want to pull your hair out, shut down and walk away. Before you do that, try submitting the following code.

```
*'; *=; *); */; %mend; run;
```

This string can be repeatedly submitted until you get the following message in your log. Once you get this message, you can continue on your merry programming way.

```
ERROR: No matching %MACRO statement for this %MEND statement.
```

FILE NOT FOUND

Few things are more frustrating than executing your code just to get to the end of the execution and find out that the file you were referencing does not exist, like realizing you're out of limes when serving margaritas. When reading in external files, it is important to check that the file exists before you get too far into the weeds. Otherwise, you can end up with the following ERROR message.

```
ERROR: Physical file does not exist, C:\Users\gonza\ISO_dates2.csv.
```

To avoid executing code on a file that does not exist, you can check for the existence of the file and only execute the code if the file is found. The use of the FILEEXIST function can be used in conditional logic to only execute the code if the file exists. If the file does exist, FILEEXIST returns a one, otherwise it returns a zero.

```
%macro filechk(infl = );
   %if %sysfunc(fileexist(&infl)) %then %do;
      data dates;
        infile "&infl" delimiter = ',' missover dsd;
        informat dat $50.; informat dtc $50.;
        input dat $ dtc $;
      run;
   %end;
   %else %do;
      %put ERROR: &infl does not exist;
   %end;
%mend filechk;
%filechk(infl = C:\Users\gonza\ISO_dates.csv)
```

Macro 1: Fileexist

If you are reading in the external file, using a LIBNAME statement, instead of FILEEXIST function, you could use the LIBREF function to check that the library reference was assigned successfully. With LIBREF if the library reference is successfully assigned it returns a zero, otherwise it returns a non-zero.

```
%macro filechk(infl = );
   libname mylib xlsx "&infl";
   %if %sysfunc(fileexist(&infl)) %then %do;
      data dates;
        set mylib.Dates;
      run;
   %end;
   %else %do;
      %put %sysfunc(compress(ERR OR:)) &infl does not exist;
   %end;
%mend filechk;
%filechk(infl = C:\Users\gonza\ISO_dates.xlsx)
```

Macro 2: Libref

CONCLUSION

Debugging SAS code may often feel like taming a wild beast, but with the insights and tools discussed in this paper, you can tackle it with confidence and skill. From the powerful Debugger in SAS Display Manager and SAS Enterprise Guide to the versatile PUTLOG and PUT statements, and macro language debugging options like MPRINT, MLOGIC, and SYMBOLGEN, you now have a full arsenal of techniques at your fingertips. Whether you're taking a tried-and-true cocktail recipe or fine-tuning your own creations, these strategies can turn frustration into triumph.

Remember, every error message is just a puzzle waiting to be solved, and with a little patience (and maybe a stiff drink), you'll emerge victorious. Debugging isn't just a chore—it's a superpower that equips you to write cleaner, smarter, and more robust SAS programs. So, the next time your code throws a tantrum, you'll know exactly how to calm it down (with a nice adult beverage). (Wilson, 2011) (Tricia Aanderud, 2014)

REFERENCES AND RECOMMENDED READINGS

Burlew, M. M. (2001). *Debugging SAS® Programs A Handbook of Tools and Techniques* . Cary, NC: SAS Institute, Inc.

Common Error Messages in SAS | SAS Learning Modules. UCLA: Statistical Consulting Group. (n.d.). Retrieved Mar 2025, from https://stats.oarc.ucla.edu/sas/modules/common-error-messages-in-sas/

Hemedinger, C. (2016, Nov 30). Using the DATA step debugger in SAS Enterprise Guide. Retrieved Mar 2025, from https://blogs.sas.com/content/sasdummy/2016/11/30/data-step-debugger-sas-eg/

SAS Communities. (2020, Aug). Difference between PUTLOG and PUT statements. Retrieved Mar 2025, from https://communities.sas.com/t5/SAS-Certification/Difference-between-PUTLOG-and-PUT-statements/td-p/674610

SAS Institute, Inc. (2017, Dec 6). *SAS® 9.4 Global Statements: Reference.* Cary, NC: SAS Institute, Inc. Retrieved Mar 2025, from https://documentation.sas.com/api/collections/pgmsascdc/9.4_3.3/docsets/lestmtsglobal/content/lestmtsglobal.pdf

Tricia Aanderud, A. H. (2014). Uncover the Most Common SAS Stored Process Errors. https://support.sas.com/resources/papers/proceedings14/1245-2014.pdf.

Wicklin, R. (2023, Jan 23). Use the PUTLOG Statement to Write Errors, Warnings, and Notes to the SAS Log. Retrieved Oct 2024, from https://blogs.sas.com/content/iml/2023/01/23/putlog-statement.html

Wilson, K. (2011). The Top 10 Head-Scratchers: SAS Log Messages That Prompt a Call to SAS Technial Support. *NESUG.* https://www.lexjansen.com/nesug/nesug11/ds/ds13.pdf.

ACKNOWLEDGMENTS

The authors want to thank Lex Jansen for continuing to publish SAS Proceedings from 1976 to present. The website searches 36,515 papers (as of the date of writing this paper) and is a wealth of information for SAS Users. https://www.lexjansen.com.

We acknowledge the use of Microsoft Copilot for assisting in crafting and refining the witty cocktail-themed elements of this paper. The technical content and research remain entirely our own.

CONTACT INFORMATION

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

Lisa Mendez
Army MEDCOM
sasebmendez@gmail.com

Richann Jean Watson
DataRich Consulting
richann.watson@datarichconsulting.com