Paper RF-12-2013

The Utility of the DATA Step Debugger in Logic Errors

Darryl Nousome, MPH, University of Southern California, Los Angeles, CA

ABSTRACT

There are a few strategies in debugging errors that arise during SAS® programming. Errors in programming can manifest as either syntax or logic errors. Syntax errors tend to be easily identified as they stop the program and generate error messages in the log window. While logic errors will not halt SAS during the DATA step compilation phase, this may result in data that is unintended. A useful tool in examining logic errors is invoking the DATA Step Debugger, which allows viewing of the Program Data Vector (PDV) during the execution of the DATA step. When the DATA Step Debugger is running, there are many options that can examine values of selected variables, suspend statements, display the values of any variables in the PDV, or other commands. This process can aid in identifying segments of code that may contain logic errors. This paper will highlight some techniques that are used by the DATA Step Debugger.

INTRODUCTION

Logic errors executed during the DATA step compilation phase will not stop SAS during the compilation phase, but the error can generate unwanted output. These errors can arise when programmers overlook what is being inputted by the PDV during a DATA step. The DATA Step Debugger allows the programmer to debug SAS DATA steps by adding the DEBUG option to the end of a DATA statement. During the DATA Step Debugger, SAS will first compile the DATA step, display the DEBUG windows, and interactively allow viewing and updating of the PDV using certain commands.

APPLICATION OF THE DATA STEP DEBUGGER

The following application uses a program (Program 1) to reshape the Radiation data (Table 1) that is currently stored in a long format to be placed into wide format, the Survival data set (Table 2), in preparation for survival analysis. Each participant in the study has received a different type of radiation at three different time points in the study.

Program 1. Program Containing Logic Error

```
data wide (drop=time radiation); set radiation;
             by id;
       retain rad1-rad3;
       if time=1 then rad1=radiation;
             else if time=2 then rad2=radiation;
             else rad3=radiation;
       if last.id;
run;
```

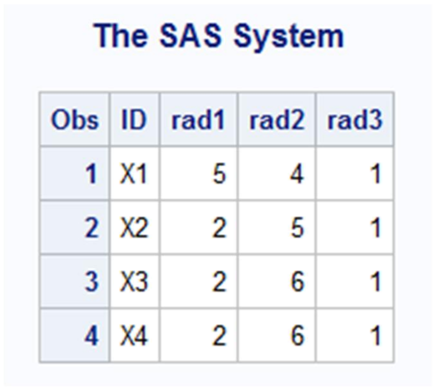
ID	Time	Radiation
X1	1	5
X1	2	4
X1	3	1
X2	1	2
X2	2	5
X3	2	6
X4	3	1

Table 1. Radiation data

ID	Rad1	Rad2	Rad3
X1	5	4	1
X2	2	5	.
X3	.	6	.
X4	.	.	1

Table 2. Survival data

Program 1 is the initial program to attempt to reshape the data, but this Program results in the values for each radiation value carried down for all individuals. For example, as shown in Output 1, ID X3 should be missing on the first radiation treatment time point, but its value appears to be carried down from ID X2.



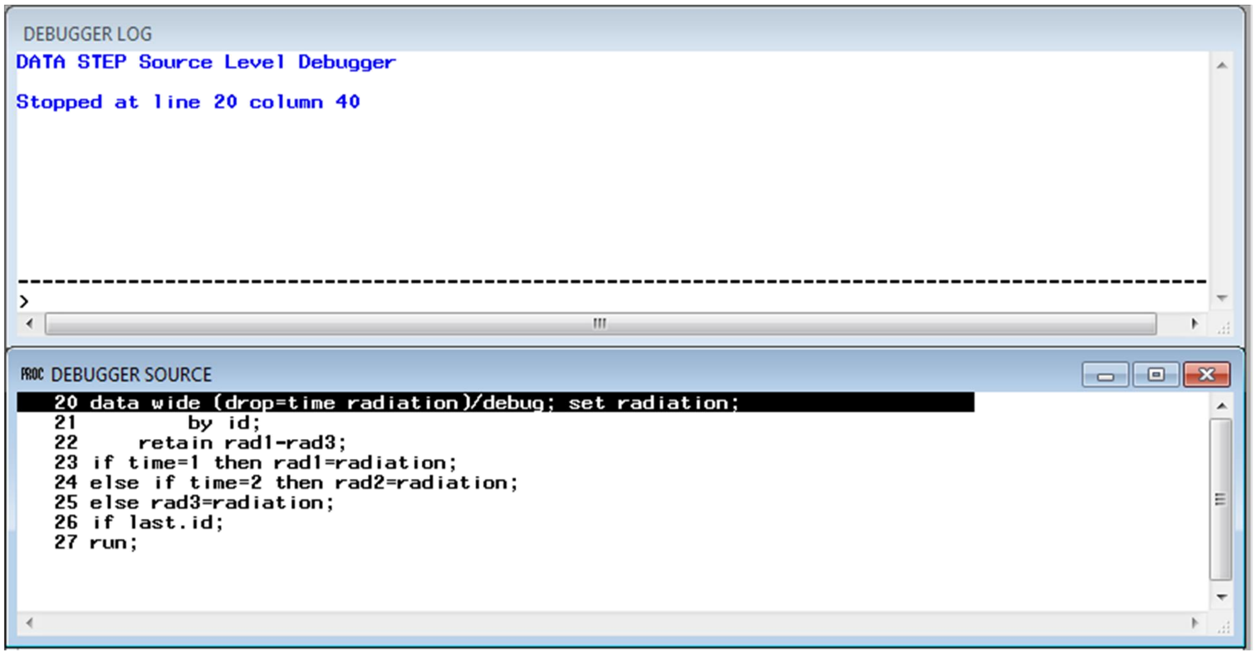
The SAS System

Obs	ID	rad1	rad2	rad3
1	X1	5	4	1
2	X2	2	5	1
3	X3	2	6	1
4	X4	2	6	1

Output 1. Undesired Output from Program 1

This logic error does not stop the SAS program, but it does not produce the desired output. The DATA Step Debugger can be initialized here. By adding a DEBUG option after the DATA statement and rerunning the statement, the debugger will be invoked.

Two windows appear: the DEBUGGER LOG and the DEBUGGER SOURCE.



```
DATA STEP Source Level Debugger

Stopped at line 20 column 40
```

```
20 data wide (drop=time radiation)/debug; set radiation;
21         by id;
22     retain rad1-rad3;
23 if time=1 then rad1=radiation;
24 else if time=2 then rad2=radiation;
25 else rad3=radiation;
26 if last.id;
27 run;
```

Window 1. DEBUGGER LOG and DEBUGGER SOURCE windows

The DEBUGGER LOG window highlights current program that is running. DEBUG commands can be entered directly into the LOG window in the command line. The command line is below the row of dashes (Window 1).

The DEBUGGER SOURCE window holds the current DATA step program. The lines correspond to the lines of the current program in both the LOG and DEBUGGER LOG windows. The line to be executed by SAS is highlighted within the window, and in this instance, before line 20. All iterations will be run within the DEBUGGER SOURCE and LOG as each line is initialized and entered into the PDV.

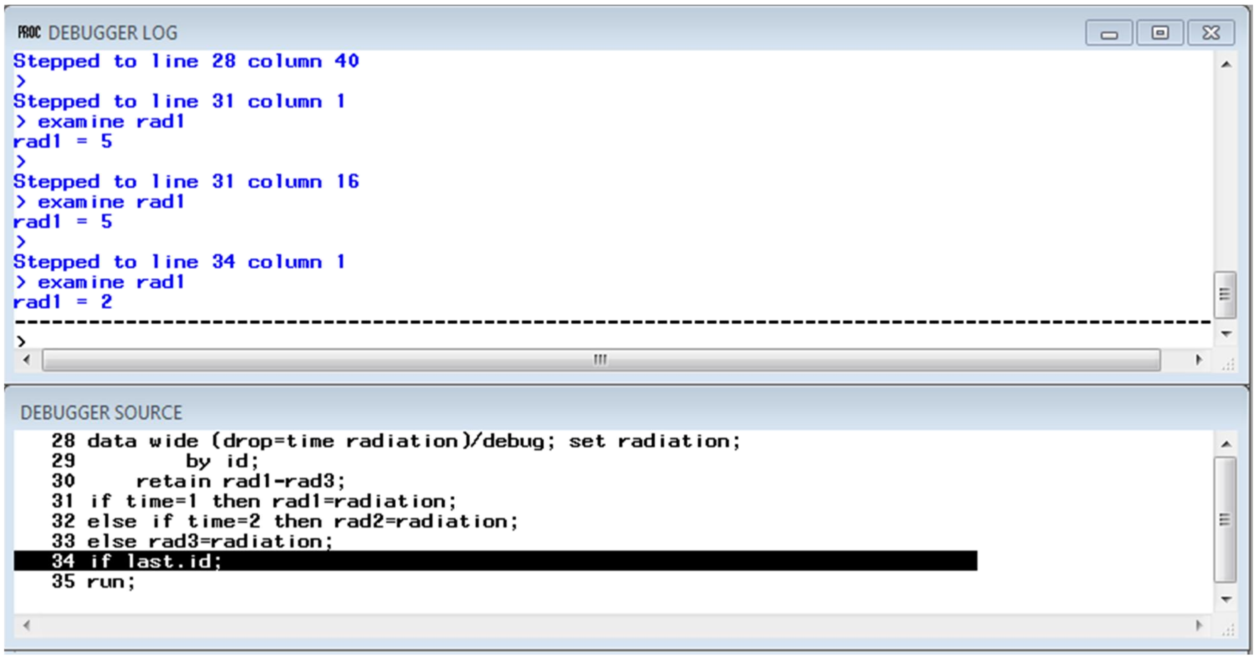
DEBUGGER COMMANDS

Various DEBUGGER commands are highlighted below with commands and their shortcuts (in brackets) to be used in the DEBUGGER LOG window.

EXAMINE [e]

The EXAMINE command displays the current value of the variable in the PDV. The SAS keyword _all_ can be used to display all variables. Because we are interested in the error in input of the rad1 variable, that unique variable name can be entered as well (Window 2).

```
Examine _all_
```



```
Stepped to line 28 column 40
>
Stepped to line 31 column 1
> examine rad1
rad1 = 5
>
Stepped to line 31 column 16
> examine rad1
rad1 = 5
>
Stepped to line 34 column 1
> examine rad1
rad1 = 2
-------------------------------------------------------------------------------------------
>
```

```
28 data wide (drop=time radiation)/debug; set radiation;
29           by id;
30      retain rad1-rad3;
31 if time=1 then rad1=radiation;
32 else if time=2 then rad2=radiation;
33 else rad3=radiation;
34 if last.id;
35 run;
```

Window 2. Application of the EXAMINE command

STEP [st]

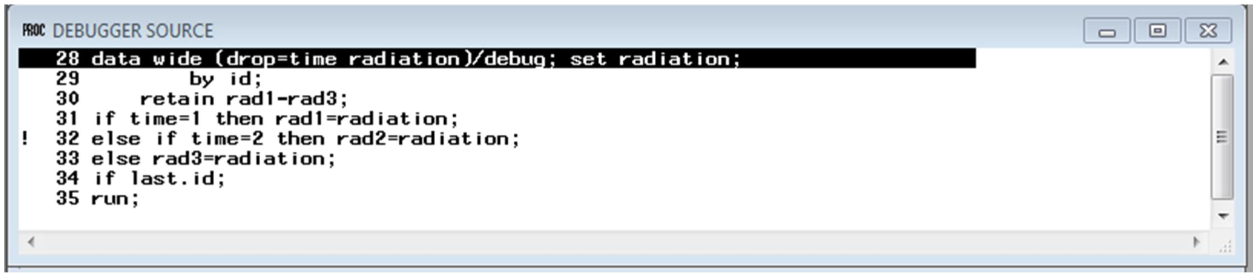
The STEP command executes each line where the DATA Step Debugger pointer is. By default, the ENTER key also functions as the STEP command. The STEP command can be combined with a value to enter the command n times.

```
step 3
```

BREAK [b]

The BREAK command suspends the DATA step with a certain condition. For example, setting a breakpoint can be done at a certain line. Multiple BREAK commands can be done in one DEBUG session. Here the BREAK point is set at line 32, where an exclamation point is shown in Window 3.

```
break 32
```



```
   28 data wide (drop=time radiation)/debug; set radiation;
   29           by id;
   30      retain rad1-rad3;
   31 if time=1 then rad1=radiation;
!  32 else if time=2 then rad2=radiation;
   33 else rad3=radiation;
   34 if last.id;
   35 run;
```

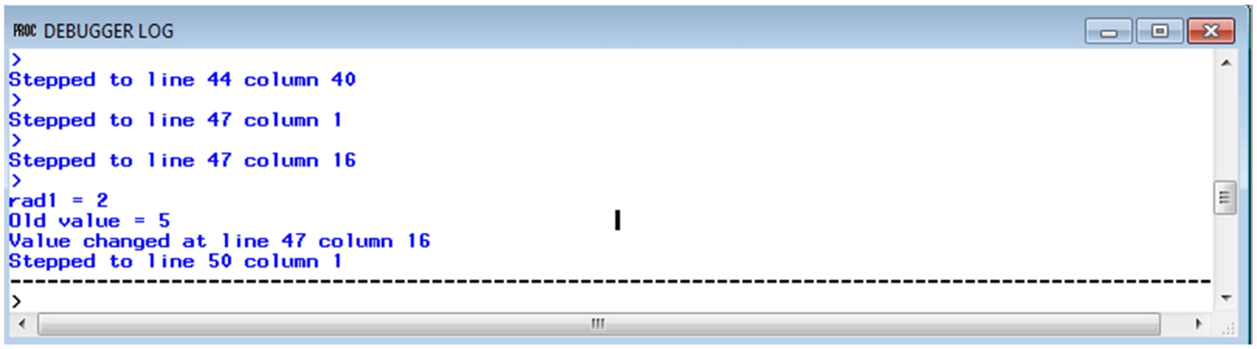
Window 3. BREAK point entered into line 32

More advanced commands including conditional statements like WHEN and group processing like DO. Here this BREAK command with DO and WHEN statements will examine ID X3 at line 32 to determine which value will be read by the PDV at the time.

```
break 32 when ID="X3" do; examine _all_; end;
```

WATCH [w]
The WATCH command displays both the old and new value for the specified variable that will be entered into the PDV (Window 4). For each variable to under the WATCH command, the DEBUGGER LOG window will show changes.

```
watch rad1
```



```
>
Stepped to line 44 column 40
>
Stepped to line 47 column 1
>
Stepped to line 47 column 16
>
rad1 = 2
Old value = 5
Value changed at line 47 column 16
Stepped to line 50 column 1
--------------------------------------------------------------------------------
>
```

Window 4. WATCH command used to observe changes in variable rad1

GO [g]
The GO command will continue to execute the DATA steps until it reaches the specified BREAK point, WATCH points, or the completion of the DATA step if none of the previous commands are specified. The GO command issued with no line number will automatically run to the next BREAK or WATCH point. If a line number is entered, the debugger will run until it can reach the line, including running subsequent iterations.

TRACE [t]
This command displays all commands run in the DEBUGGER LOG window. The TRACE command needs to be initialized to work (the default is off).

```
trace on
```

When used in conjunction with GO and BREAK commands, it will display every continuous record and not only the most recent command executed.

DELETE [d]
With the DELETE command, unwanted break points and watched variables can be deleted. The following command will delete the break point set at line 32.

```
delete break 32
```

SET
This will allow for manually setting values that are not initially part of the program submitted to the PDV. The current iteration the run by the DATA Step Debugger will be the value changed by the SET command.

```
set rad1=3
```

LIST [l]
The LIST command lists all BREAK points, variables under WATCH, files, and data sets used in the DATA Step Debugger. The variable list _all_ can explore all commands at once.

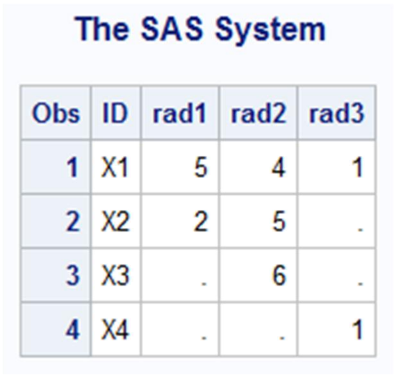
QUIT [q]
This ends the DATA Step Debugger and terminates the program. The DATA STEP program will stay open even if the program has completed running. This will reopen the original SAS program.

CORRECTING THE DATA STEP LOGIC ERROR

After exploring the program using the DATA Step Debugger, specifically the WATCH and EXAMINE commands, it appeared that upon the iteration of the program for individual X3, the value was retained in the PDV from the previous individual. Therefore, a new program (Program 2) was written to initialize each new rad variable that was created to missing by each individual. This produces the desired output (Output 2).

Program 2. Fixed program

```
data wide (drop=time radiation); set radiation;
       by id;
       retain rad1-rad3;
       if first.id then do;
              rad1=.; rad2=.; rad3=.;
       end;
       if time=1 then rad1=radiation;
              else if time=2 then rad2=radiation;
              else rad3=radiation;
       if last.id;
run;
```



The SAS System

Obs	ID	rad1	rad2	rad3
1	X1	5	4	1
2	X2	2	5	.
3	X3	.	6	.
4	X4	.	.	1

Output 2. Corrected Output after Identifying Logic Error

CONCLUSION

Using the DATA Step Debugger can help to alleviate problems due to logic errors. Because logic errors do not stop the SAS program from compiling, the Data Step Debugger is an invaluable tool in diagnosing and eliminating logic errors that can arise.

ACKNOWLEDGEMENTS

The author would like to acknowledge the assistance of Arthur Li.

REFERENCES

1. Li, Arthur. Handbook of SAS® DATA Step Programming. Chapman and Hall, 2013.

CONTACT INFORMATION

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

Darryl Nousome
University of Southern California
2001 N Soto Street #67
Los Angeles, CA 90032
nousome@usc.edu