PharmaSUG 2012 – Paper CC23

Figure Out Your Programming Logic Errors via DATA Step Debugger

Wuchen Zhao, University of Southern California, Los Angeles, CA

ABSTRACT

Programming errors can be categorized into syntax errors and logic errors. Syntax errors can be easily fixed because these types of errors can be detected by SAS® during the DATA step compilation phase and SAS will stop the program to generate an error report in the log window. Unlike syntax errors, logic errors are difficult to be found and/or corrected since programs with logic errors often operate correctly but result in unexpected results. One of the useful tools to repair logic errors is to utilize DATA Step Debugger, which allows you to see the contents of the Program Data Vector (PDV) during the DATA step execution. With the debugger running, you will be able to monitor and update data values by using data debugging commands which will help you to understand where the problem is so that you will be able to fix the error. This paper will illustrate some debugging techniques by using DATA Step Debugger.

INTRODUCTION

A logic error, sometimes called a semantic error, is an error that does not terminate your program abnormally; however, the error will generate unexpected output. This type of error is a programmer's worst nightmare. Logic errors in the SAS language are often related to understanding how the DATA step and PDV work during the DATA step execution phase. Logic errors can be corrected by reviewing and inspecting the contents of the PDV during each iteration of the execution phase. One way to view the contents in the PDV is to use the PUT _ALL_ statement in the program, which will result in SAS listing the PDV contents in the SAS log. A better approach is to invoke DATA Step Debugger (DSD) by adding the DEBUG option to the DATA statement. Once the DATA step is invoked, you will be able to see the PDV contents and update data values interactively by entering DATA debugging commends.

SITUATIONS FOR USING DSD

To illustrate the use of DSD, consider the data set *example* (Table 1). Suppose that there are two persons, A and B. Person A has 5 measurements of SBP and person B has 3 measurements of SBP; they are listed in time sequence. Now suppose you would like to carry down the non-missing data in order to replace the missing values for each person. The resulting data should be like the one in *Result* (Table 2).

Name	Time	SBP
A	1	95
A	2	.
A	3	110
A	4	110
A	5	.
B	1	.
B	2	90
B	3	.

Table 1. Data of *Example*

Name	Time	SBP
A	1	95
A	2	95
A	3	110
A	4	110
A	5	110
B	1	.
B	2	90
B	3	90

Table 2. Data of *Result*

Program below is the first attempt to carry down the non-missing data but the resulting data is not quite correct because the value of SBP at the 1st time for person B is equal to 110, which is the value from the previous observation.

```
data wrong (drop = foo);
       set example;
       retain foo;
       if not missing(sbp) then foo=sbp;
              else if missing(sbp) then sbp=foo;
run;
```

```
The SAS System
Obs    name    time    sbp
 1      A        1      95
 2      A        2      95
```

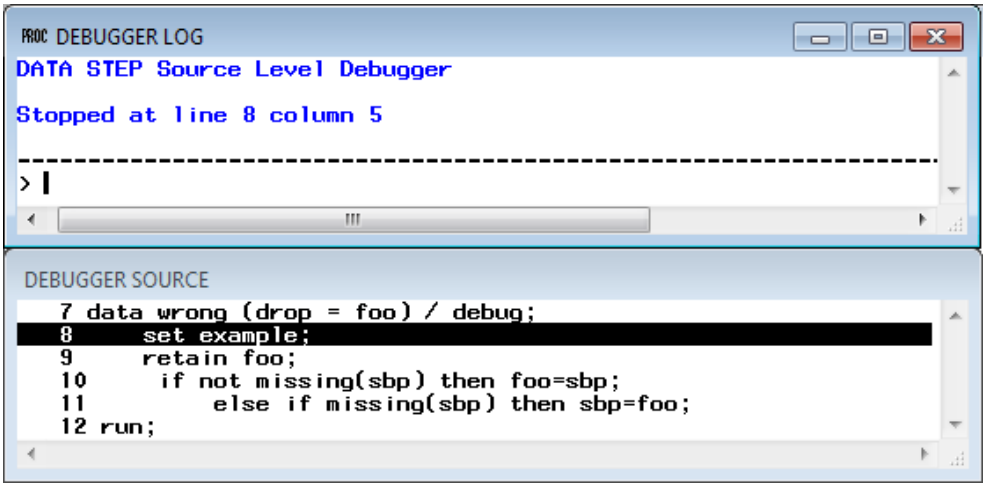
3	A	3	110
4	A	4	110
5	A	5	110
6	B	1	110
7	B	2	90
8	B	3	90

Output 1. The Data Set *Wrong* generated after the first attempt

DATA Step Debugger can be used to fix the logic error in Output 1 from above. To invoke the DSD, you need to add the DEBUG option in the DATA statement and resubmit the entire DATA step as shown in program below.

```
data wrong (drop = foo) / debug;
      set example;
      retain foo;
      if not missing(sbp) then foo=sbp;
            else if missing(sbp) then sbp=foo;
run;
```

Once the debugger is invoked, a DEBUGGER LOG window and a DEBUGGER SOURCE window will pop- up as the ones shown in Display 1.



Display 1. The DEBUGGER LOG window and the DEBUGGER SOURCE window

To operate the debugging session, type DSD commands on the debugger command line in the bottom of the DEBUGGER LOG window under the dash line and press the Enter button. The DSD commands that have been issued will be displayed along with the results in this window above the dash line.

The DATA step program with the DEBUG option is displayed in the DEBUGGER SOURCE window. The line numbers in the DEBUGGER SOURCE window are consistent with the line numbers in the SAS log. The first executable line (line 8) is highlighted in reverse color, which means that the pointer of the DSD is now at this line. From the DEBUGGER LOG window you can see that the pointer stopped at line 8 column 5; the DSD will execute statements or perform other operations from this location of the program.

The basic and most important manipulation in a DSD session is to order DSD to execute statements in the DATA step program. You can use STEP commands or press the Enter key to execute the DATA step program within DSD.

STEP <*n*>
Alias: st
Example: st 5 /* Execute 5 statements, equivalent to pressing Enter key 5 times */

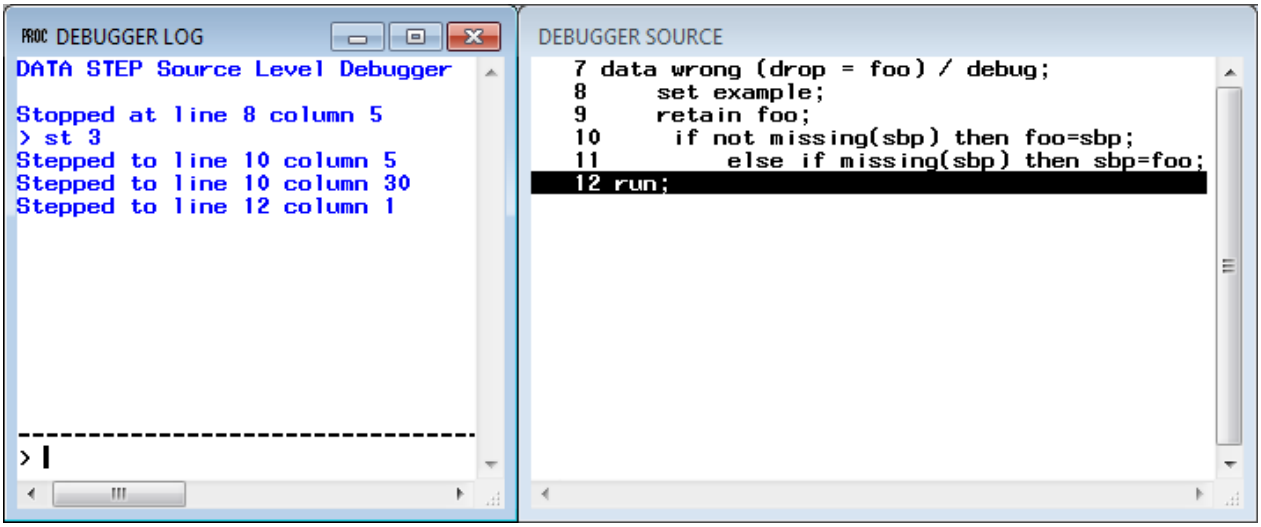
A STEP command executes statements from the location where the DSD pointer stops.

A single STEP command executes only one executable statement and moves the DSD pointer to the beginning of the next executable statement. You can assign DSD to execute more than one statement by

adding a number after the STEP command. A STEP command without any following numbers is equivalent to pressing the Enter key once.

Use the STEP command to execute 3 statements; the pointer stops at line 12 column 1 (Display 2):

st 3



Display 2. The DATA Step Debugger executes 3 statements

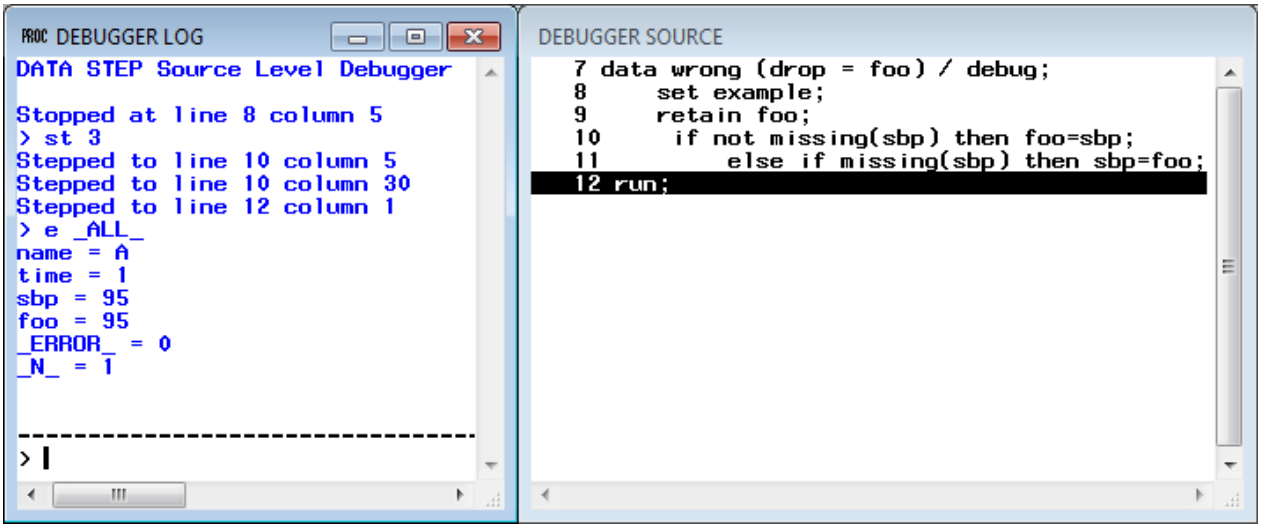
During the operation of the three executable statements, the first observation was read and FOO was set-up to be equal to SBP, which is 95. To access the PDV and to examine the values of each variable, you can use the EXAMINE command.

EXAMINE *variable-1 <format-1> <...variable-n <format-n>>*
EXAMINE _ALL_ *<format>*
Alias: e
Example: e _ALL_

An EXAMINE command displays the values of specified variables in the DEBUGGER LOG window. Use the _ALL_ option after EXAMINE to display the values of all the variables.

Use the EXAMINE command to display all the values of variables at the present time (Display 3):

e _ALL_



Display 3. Examine and display all the variables

In this example, the logic error may occur around the 5th iteration. It is difficult to tell how many steps need to be executed before reaching the 5th iteration and repeatedly pressing the Enter key is impractical. In order to solve this problem, the BREAK and GO commands can be used to complete these operations.

BREAK *location* <AFTER *count*> <WHEN *expression*> <DO *group* >
Alias: b
Example: b 25 when (sbp=.) do;if time=1 then e name;else st;end;

A BREAK command suspends DATA step execution at the specified location.

The location argument should be either the line number or an asterisk (*). The asterisk represents the current line. An AFTER argument suspends the execution when the statement has been executed every *count* times. This option is very useful for the DO loop. A WHEN option suspends the execution when the *expression* is true.

The effect and usage of the DO option are similar to the DO statement in the DATA step. You can assign multiple manipulations in a DO group. The syntax of the DO group is listed below:

DO; ***command-1*** < ... ; ***command-n***; >END;

Commands in DO groups are as the following:

IF ***expression*** THEN ***command***; <ELSE ***command***;>

IF ***expression*** THEN DO ***group***; <ELSE DO ***group***;>

Notice that the statements in DO groups need to be DSD commands.

GO <*line-number* | *label*>
Alias: g
Example: g 25 /* Go to Line 25 */

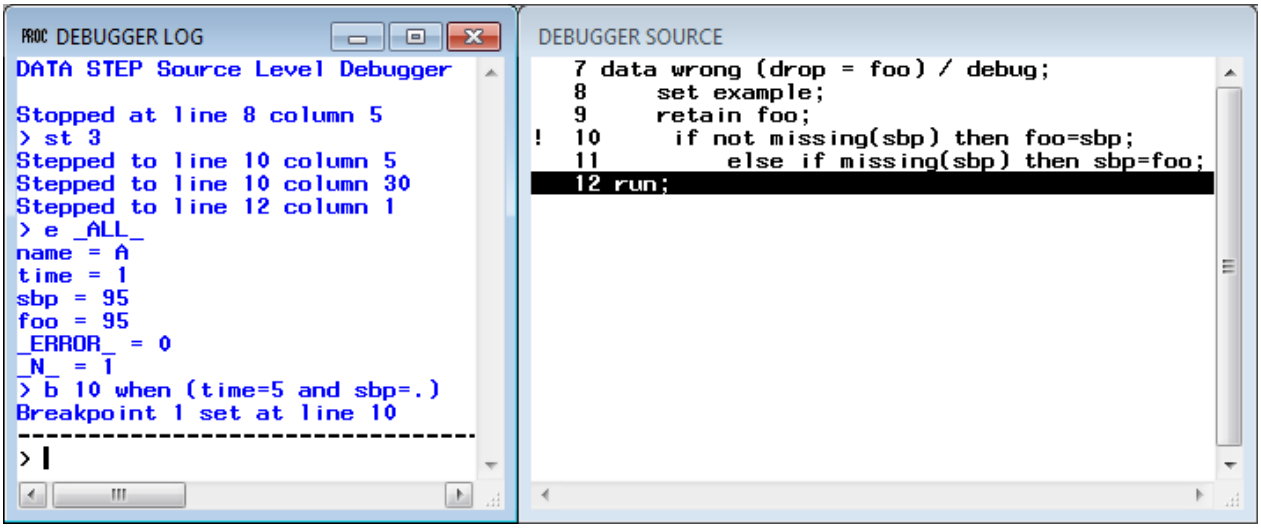
A GO command will execute lines continuously until the conditions meet certain criteria.

The execution of a GO command will stop when one of the following conditions occurs: the execution reaches the *line-number* (which has been specified in the arguments), the breakpoint is encountered, the values of any watched variables change (see reference in Table 3), the DATA STEP program has completed all the execution phases.

Use the BREAK command to set up a breakpoint in the 5th iteration:

b 10 when (time=5 and sbp=.)

After submitting the BREAK command, an exclamation mark (!) will appear in front of the line that has just been specified in the DEBUGGER SOURCE window (Display 4), which indicates that a breakpoint has been set-up successfully.



Display 4. Set up a breakpoint

Type a GO command to run the DATA step until the breakpoint occurs:

g

In order to avoid unexpected results and/or difficulties, a DELETE command can be used to delete the breakpoint.

DELETE BREAK *location*
DELETE WATCH *variable(s)* | _ALL_
Alias: d
Example: d b 5 /* Delete breakpoint on line 5 */

There are two arguments that follow the DELETE command: BREAK (b) and WATCH (w).

To delete a breakpoint at a certain line, you need to specify the line number after the BREAK argument. You can use an asterisk (*) or _ALL_ instead of the line number after the BREAK argument, which means deleting the breakpoint from the current line or deleting all the breakpoints in the DSD session.

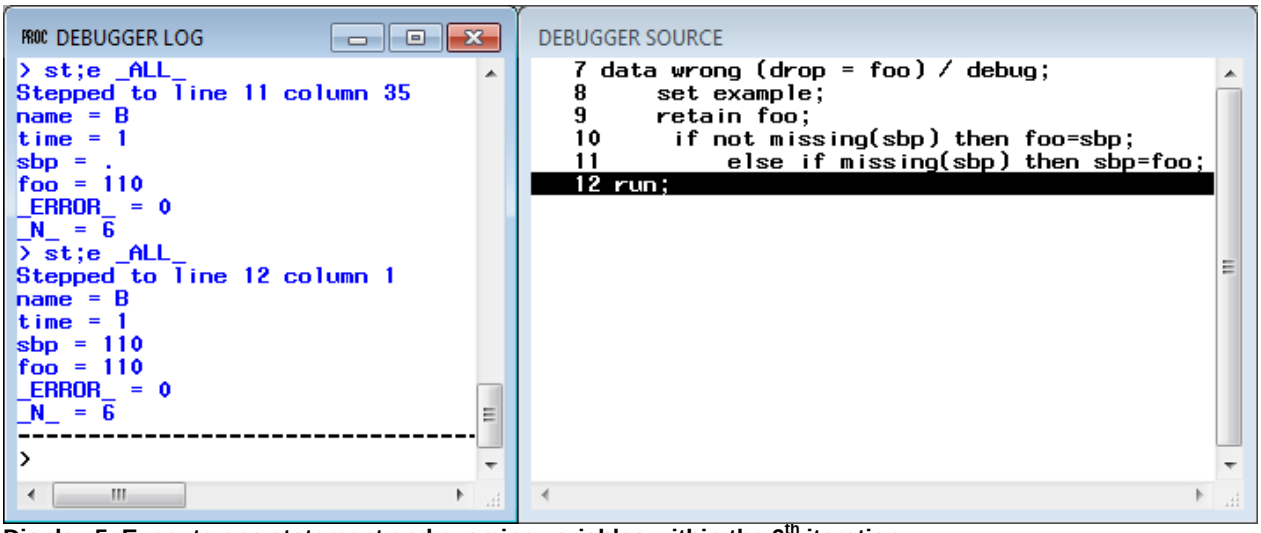
You can specify any variable names or type _ALL_ after the WATCH argument to delete variable(s) being watched (See reference in Table 1).

Use the DELETE command to delete the break point in line 10:

d b 10

Since the logic error in this example occurs around the 5th iteration, you may want to execute statements one at a time. Use semicolons to separate commands when entering multiple commands at one line:

st;e _ALL_



Display 5. Execute one statement and examine variables within the 6th iteration

Display 5 illustrates where the logic error occurs. In the DEBUGGER LOG above, the contents of the PDV is listed after the 'ST; E _ALL_' commands. Notice that the value of SBP is missing after the first 'ST; E _ALL_' command but SBP is assigned to 110 after the second 'ST; E _ALL_' command. The above shows that the value in SBP for the first observation for person B is carried from the last observation from person A. If you set the FOO to missing before assigning FOO to SBP when reading the first observation of each subject, the problem will be solved.

After discovering the crux of the error, press F8 or use the QUIT command to terminate the DSD session.

QUIT
Alias: q

Terminate a DSD session.

Based on the deduction above, one statement is added to correct the program by resetting FOO to missing when

executing the first observation of each subject (see program below).

```
data right (drop = foo);
       set example;
       retain foo;
       if time=1 then foo=.;
       if not missing(sbp) then foo=sbp;
              else if missing(sbp) then sbp=foo;
run;
```

```
The SAS System
Obs    name    time    sbp
 1      A        1      95
 2      A        2      95
 3      A        3     110
 4      A        4     110
 5      A        5     110
 6      B        1       .
 7      B        2      90
 8      B        3      90
```

Output 2. The Data Set *Right* after correcting the DATA step program

COMMANDS TO OPERATE THE DATA SET DEBUGGER

Beside the DSD commands that have been introduced in the example above, there are some other frequently-used commands to manipulate the debugger. Some useful DSD commands are listed below in Table 3:

Controlling Program Execution	JUMP	**JUMP** *line-number* | *label* Alias: j Example: j 25 /* Jump to Line 25 without executing the intervening statements */ A JUMP command moves the pointer of execution to the beginning of the specified line. The intervening statements will not be executed if the specified line is after the current line. If the specified line is before the current line, then the DSD will not undo the intervening statements but instead simply moves the pointer to the line that was specified.
Manipulating DATA Step Variables	CALCULATE	**CALC** *expression* Alias: None Example: calc sbp*5 CALCULATE is a command to evaluate expressions and to display results in the DSD.
	DESCRIBE	**DESCRIBE** *variable(s)* | _ALL_ Alias: desc Example: desc sbp DESCRIBE is a command to display the attributes of specified variable(s), including the name, type, length, as well as the informat, format, and label (if they exist).
	SET	**SET** *variable=expression* Alias: None Example: set time = time+1 A SET command assigns a new value to the specified variable.
Manipulating Debugging Requests	LIST	**LIST** _ALL_ | BREAK | DATASETS | FILES | INFILES | WATCH Alias: l Example: l b w A LIST command displays information of the DSD session.

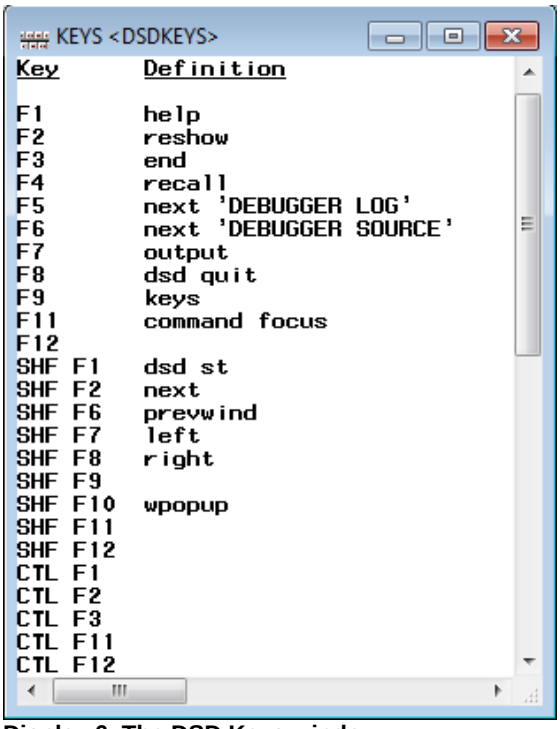
		The arguments BREAK and WATCH can be used to display information of breakpoints and watched variables. The DATASETS argument displays all the data sets that are being used. The FILES and INFILES arguments provide the information of all external data sets that the current DATA step writes to and reads from. You can use '_ALL' to acquire all the information.
	WATCH	**WATCH** *variable(s)* Alias: w Example: w foo A WATCH command monitors the variable(s) that have been specified. The debugger suspends execution when the value of a watched variable changes. The DSD displays the current value and the previous value of that variable in the DEBUGGER LOG window, the locations of where the value changes, and where the pointer stops.

Table 3. Commands of the DATA Step Debugger

DEFINE FUNCTION KEYS

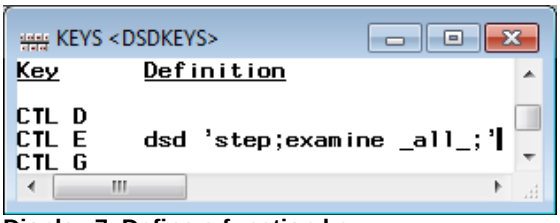
Sometimes you may need to repeat one or multiple commands many times; it is very inconvenient to type the command again and again. In this situation, you can assign commands to a specific key in the DSD Keys window (see Display 6) to simplify your manipulations.

Press F9 to invoke the DSD Keys window when the DSD is active or find the Keys window from the SAS menu bar following these steps: Tools -> Options -> Keys.



Display 6. The DSD Keys window

The DSD Keys window consists of two parts: Key and Definition. Keys that are available are listed on the left in the Keys window. To define a function key, you need to move your cursor to the corresponding Definition column, type in the command with the term 'DSD' at the beginning, and then close the Keys window. To assign more than one command to a function key, you need to enclose the commands with quotation marks and separate commands by semicolons (;). Display 7 is an example of defining a function key, Ctrl + E, to fulfill two commands.



Display 7. Define a function key

CONCLUSION

Logic errors are common and are difficult to locate and correct, especially when the DATA step is complicated and/or the data set is enormous. With the assistance of the SAS DATA Step Debugger, it will be easier and more practical to scrutinize the DATA step and to correct any errors by inspecting each iteration. In addition, DSD is also a powerful tool to better learn the operation modes of the PDV.

CONTACT INFORMATION

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

Name: Wuchen Zhao
Enterprise: University of Southern California
Address: Division of Biostatistics, 2001 N Soto Street
City, State ZIP: Los Angeles, CA 90089-9237
E-mail: zhaowuchen@gmail.com