Paper 1797-2018

Navigating through Code with the DATA Step Debugger

Mina Chen, Roche (China) Holding Ltd.

ABSTRACT

How would you know if there is a logic error in your program? What is a good way to determine whether there is a logic error in the program? Have you ever run an intricate DATA step and the results are not as you expect? Viewing the SAS log will not help you debug the program because the data are valid and no errors appear in the log. The DATA Step Debugger (DSD) in SAS® Enterprise Guide provides a nice interactive way to watch what's going on in the DATA step and quickly identify data and logic errors. It allows you to control the execution of a DATA step program, step through your program line by line, or suspend execution of your program at selected statements. It not only allows you to watch code execute, but also allows you to change the values manually in the program data vector as your program is running.

The DATA Step Debugger is a useful tool for all SAS users, from the beginner to advanced programmer. This paper will demonstrate how to use the new DATA Step Debugger in SAS® Enterprise Guide to identify logic errors even in the most complex data steps.

INTRODUCTION

Any SAS programmer who has worked with a DATA step program must know that there is a chance that things will not go as planned. The problems usually fall into two kinds of errors: syntax error or logic error. Syntax error will prevent the DATA step from executing, so SAS log can be very helpful in finding the problem if your code has syntax errors. However, what if your code is technically reasonable, but is not producing the results as you expect? This is so called logic error. This type of error will not terminate your program abnormally and display error message in SAS log, but will generate unexpected results.

So what is a good way to determine whether there is a logic error in the program? The DATA Step Debugger introduced since SAS Enterprise Guide 7.13 provides a nice interactive way to see what's going on within the DATA step. By using the DATA Step Debugger, you are able to view the contents in the Program Data Vector (PDV), issue debugger commands, and update data values interactively.

This paper is intended to provide an overview of the DATA Step Debugger in SAS Enterprise Guide 7.15, introduce the debugger interface and debugger commands, and demonstrate its usefulness on debugging logic errors. Some examples will be provided here to show how to make use of DATA Step Debugger to solve problems in different situations.

OVERVIEW OF THE DATA STEP DEBUGGER

WHAT IS THE DATA STEP DEBUGGER?

The DATA Step Debugger in SAS Enterprise Guide provides a new debugging environment to help identify logic errors, which was introduced since SAS Enterprise Guide 7.13. It consists of user-friendly debugger windows with toolbars and a group of debugger commands that allow you to control its behavior in an interactive way.
By interacting with toolbars in debugger window or issuing debugger commands, you can execute DATA step statements one by one and pause to display the resulting variable values in a watch window. By observing the results that are displayed, you can determine where the logic error lies. Because the debugger is interactive, you can repeat the process of issuing commands and observing the results as many times as needed in a single debugging session.

In general, the DATA Step Debugger allows you to:

- control the execution of a DATA step program,
- step through your program line by line, or suspend execution of your program at selected statements.

- watch code execute and change the values manually in the PDV
- display the values of variables and assign new values to them
- display the attributes of variables

INVOKING THE DATA STEP DEBUGGER

To invoke the debugger, you need to enable the feature from the options dialog box first by pressing **Program** > **Editor Options > Enable DATA Step debugger** from the main SAS Enterprise Guide menu. Then a 'bug' toolbar icon will appear in the program editor window (see figure 1). Click the icon to launch the debugger.

Once the debugger is activated, a green bug icon appears at the beginning of the data step in the program on the left side, with a green line covering the whole DATA step code to clearly mark the area available for debugging. Alternatively, the debugger can be activated by pressing the F5 key when the cursor is anywhere inside the DATA step.

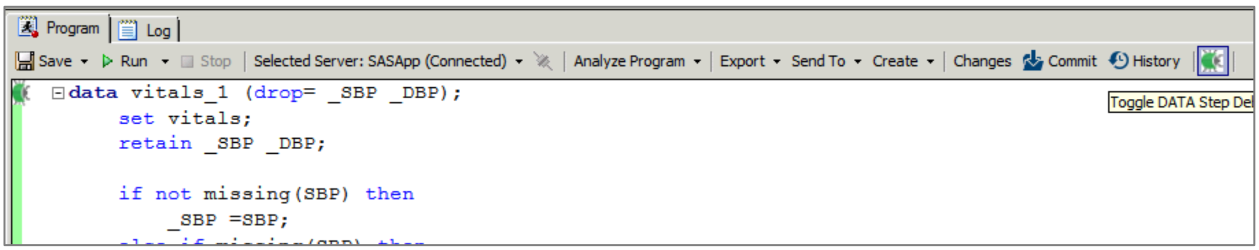


Figure 1. Debugger Toolbar Icon in the Program Editor Window

Figure 2 shows the debugger interface, which contains four parts: 1) Toolbar 2) Debug source window 3) Debug console and 4) Watch window.

The Toolbar contains five icons to enable you to start/continue debugger execution, stop debugging, step debugging, add breakpoints and clear all breakpoints/watches. See figure 2.

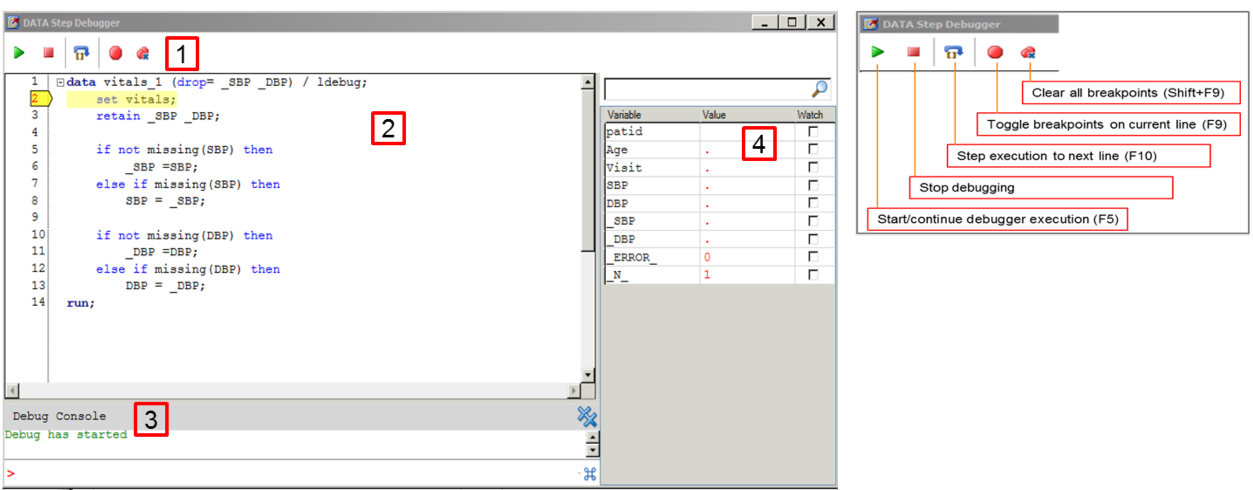


Figure 2. Data Step Debugger Interface and Toolbar Functionalities

The Debug source window displays the DATA step program with a debug option '`ldebug`' and line number added automatically. As you can see from figure 2, the first executable line (#2) is highlighted yellow and a yellow indication is drawn on this line, which means that the pointer of the DATA Step Debugger is now at this line.

The Debug console contains a command line that enables you to use debugger commands for debugging. It also displays the running log for your debug session. As you click the toolbar buttons and interact with debug source window, commands are submitted on your behalf and displayed in debug console automatically.

The Watch window provides a convenient way to show the variables in the data set, content of the PDV and check boxes to watch and monitor value changes. With every step, the watch window is updated with the latest values of the variables in your step. When a variable changes value, it's colored red. If you want the DATA step to break processing when a certain variable changes value, check the watch box for that variable.

DEBUGGER COMMANDS

As mentioned above, you can either interact with toolbar buttons or use debugger commands to control the execution of the DATA step. This section will introduce some of the most frequently used commands available with the debugger, which can fall into the following categories:

- Controlling Program Execution: **GO; STEP; JUMP**
- Manipulating DATA Step Variables: **CALCULATE; DESCRIBE; EXAMINE; SET**
- Manipulating Debugging Requests: **BREAK; DELETE; LIST; WATCH**
- Terminating the Debugger: **QUIT**

Following is a description of these commands, including alias, syntax and examples.

Controlling Program Execution:

GO (or G) <*line-number* | *label*> or press F10

This command starts or resumes execution of the DATA step. Execution continues until all observations have been read, a breakpoint specified in the GO command is reached, or a breakpoint set earlier with a BREAK command is reached.

E.g. to resume program execution and then suspend execution at the statement in line 10:

```
g 10
```

STEP (or ST)

The STEP command executes statements in the DATA step one at a time, starting with the statement at which execution was suspended.

JUMP (or J) <*line-number* | *label*>

This command moves program execution to the specified location without executing intervening statements. After executing JUMP command, you must restart execution with GO or STEP command. You can jump to any executable statement in the DATA step.

E.g. Jump to line 8:

```
j 8
```

Manipulating DATA Step Variables:

CALCULATE (or CALC) <*expression*>

The CALCULATE command evaluates debugger expressions and displays the result. The result must be numeric.

E.g. Calculate the sum of VAR1 and VAR2:

```
calc var1 + var2
```

DESCRIBE (or DESC) < *variable(s)* |_ALL_>

This command displays the attributes of one or more specified variables, including the name, type, and length of the variable, and, if present, the informat, format, or variable label.

E.g. Display the attributes of variable AGE:

```
desc age
```

Display the attributes of all variable:

```
desc _all_
```

EXAMINE (or E) *variable(s)* <*format*> |_ALL_ <*format*>

This command displays the value of one or more specified variables. The debugger displays the value using the format currently associated with the variable, unless you specify a different format.

E.g. Display the values of variables VAR1 and VAR2:

```
ex var1 var2
```

Display the SAS date variable ADT with the DATE7. format:

```
ex adt date7.
```

SET <*variable = expression*>

This command assigns a value to a specified variable. When you detect an issue during program execution, you can use this command to assign new values to variables, which enables you to continue the debugging session.

E.g. Set the variable VAR1 to the value of 4:

```
set var1=4
```

Assign to the variable SUBJECT the value ‘S’ concatenated with the previous value of SUBJECT:

```
set subject=’S’ || subject
```

<u>Manipulating Debugging Requests:</u>

BREAK (or B) *location* <AFTER *count*> <WHEN *expression*> <DO *group* >

This command suspends execution of the DATA step at a specified statement by setting a breakpoint. Following the BREAK command, you can specify a location where to set a breakpoint, and you can also add optional arguments like AFTER, WHEN expression or DO group to honor the breakpoint.

When the debugger detects a breakpoint, it will do the following:

- check the AFTER count value, if present, and suspends execution if count breakpoint activations have been reached
- evaluate the WHEN expression, if present, and suspends execution if the condition that is evaluated is true
- suspend execution if neither an AFTER nor a WHEN clause is present
- display the line number at which execution is suspended
- execute any commands that are present in a DO group

E.g. Set a breakpoint at line 6 in the current program:

```
b 6
```

Set a breakpoint at line 15 that will be honored after every third execution of line 15:

```
b 15 after 3
```

Set a breakpoint at line 15 that will be honored after every third execution of that line only when the values of both VAR1 and VAR 2 are 0:

```
b 15 after 3 when (var1=0 and var2=0)
```

Set a breakpoint at line 15 of the program and examine the values of variables VAR1 and VAR2:

```
b 15 do: ex var1 var2; end;
```

DELETE (or D) <BREAK *location*/ WATCH *variables* | _ALL_>

This command deletes breakpoints or the watch status of variables in the DATA step.

E.g. Delete the breakpoint on line 4:

```
d b 4
```

Delete all currently defined watch variables:

```
d w _all_
```

LIST (or L) <_ALL_ | BREAK | DATASETS | FILES | INFILES | WATCH>

The LIST command displays all occurrences of the item that is listed in the argument. It can display information about six types of items if they are currently defined: breakpoints (BREAK), watch variables (WATCH), external files written (FILES), external files read (INFILES), input/output data sets (DATASETS) and values of all the above (_ALL_).

E.g. List all watched variables in the current DATA step:

```
l w
```

List all breakpoints, SAS data sets, external files, and watched variables for the current DATA step:

```
l _all_
```

WATCH (or W) <*variable(s)***>**

This command specifies a variable to monitor and suspends program execution when its value changes.

Each time the value of a watched variable changes, the debugger will suspend the execution, display the line number where execution has been suspended, and display the old value and new value of the variable.

E.g. Monitor the variable AVAL for value changes:

```
w aval
```

Terminating the Debugger:

QUIT (or Q)

This command terminates the debug process.

For more details of the commands, you can refer to the DATA Step Debugger Commands instruction in Base SAS® 9.4 Utilities: Reference for a detailed explanation of each command.

DATA STEP DEBUGGER IN PRACTICE

This section will provide three program examples to demonstrate how to use DATA Step Debugger to detect logic errors. These examples are based on a source data set vitals.sas7bdat showed in figure 3, which contains the following information: patient number (patid), baseline age (Age), visit information (Visit), assessment results of systolic blood pressure (SBP) and diastolic blood pressure (DBP).

	patid	Age	Visit	SBP	DBP
1	1001	23	VISIT 1	120	80
2	1001	23	VISIT 2	119	.
3	1001	23	VISIT 3	120	78
4	1001	23	VISIT 4	.	82
5	1001	23	VISIT 5	120	80
6	1002	55	VISIT 1	188	96
7	1002	55	VISIT 2	.	90
8	1002	55	VISIT 4	182	95
9	1002	55	VISIT 5	180	.
10	1003	30	VISIT 1	200	.
11	1003	30	VISIT 2	199	99
12	1003	30	VISIT 3	190	.
13	1003	30	VISIT 4	189	90
14	1003	30	VISIT 5	190	98

Figure 3. Example Source Data (vitals.sas7bdat)

PROGRAM 1: DEBUGGING RETAIN STATEMENT

Consider the above example source data (see figure 3), suppose there are three patients 1001, 1002 and 1003. Each patient has the measurement results of SBP and DBP at different visits. However, some measurement results are missing. In order to impute the missing values, the frequently used method is to carry down the non-missing data for each person. Program below is the first attempt to carry down the non-missing data with RETAIN statement:

```
data vitals_1 (drop= _SBP _DBP);
    set vitals;
    retain _SBP _DBP;

    if not missing(SBP) then
          _SBP =SBP;
    else if missing(SBP) then
          SBP = _SBP;

    if not missing(DBP) then
          _DBP =DBP;
    else if missing(DBP) then
          DBP = _DBP;
run;
```

The code runs successfully and there are no warnings or errors in the log. However, the resulting data is not as we expected, as you can see from figure 4, the value of DBP at visit 1 for patient 1003 is equal to 95, which is the value from the measurement result of patient 1002.

	patid	Age	Visit	SBP	DBP
1	1001	23	VISIT 1	120	80
2	1001	23	VISIT 2	119	80
3	1001	23	VISIT 3	120	78
4	1001	23	VISIT 4	120	82
5	1001	23	VISIT 5	120	80
6	1002	55	VISIT 1	188	96
7	1002	55	VISIT 2	188	90
8	1002	55	VISIT 4	182	95
9	1002	55	VISIT 5	180	95
10	1003	30	VISIT 1	200	95
11	1003	30	VISIT 2	199	99
12	1003	30	VISIT 3	190	99
13	1003	30	VISIT 4	189	90
14	1003	30	VISIT 5	190	98

Figure 4. Result Data (vitals_1.sas7bdat)

Let's use the DATA step debugger to investigate the problem further. Invoke the debugger window, firstly click the STEP icon in the toolbar, you can see that the pointer stopped to line 5 column 2, as shown in the debug console. In figure 5, the watch window displays the initial variable values and PDV for patient 1001 at visit 1.

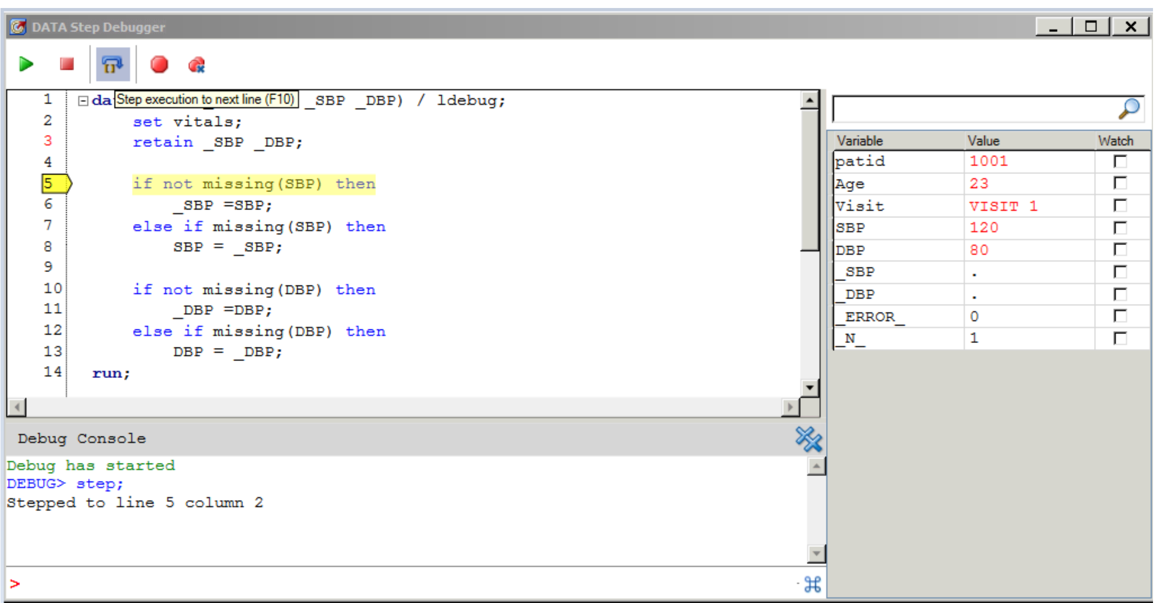


Figure 5. Debugger Session and Watch Window

In this example, the logic error may occur around the 9th iteration. It is difficult to tell how many steps need to be executed before reaching the 9th iteration and repeatedly clicking the STEP icon is impractical. In order to solve this problem, the BREAK and GO commands can be used to complete these operations. Use the BREAK command to set up a breakpoint in the 9^{th} iteration:

```
b 10 when (visit=5 and dbp=.)
```

In figure 6 you can see a red circle appears on the left side of line 10 and the whole line is highlighted red.

Click the GO icon, the debugger will execute and suspend until the WHEN condition is meet. As you can see that the variable _dbp was set to 95 at visit 5 for patient 1002, which is as we expected. So how about next? Set another breakpoint with a different condition to see what's going on at visit 1 for the next patient:

```
b 10 when (visit=1 and dbp=.)
```

Click GO and STEP icons, you can see from the watch window that the variable _DBP did not changed since last execution, and the DBP was set to 95.

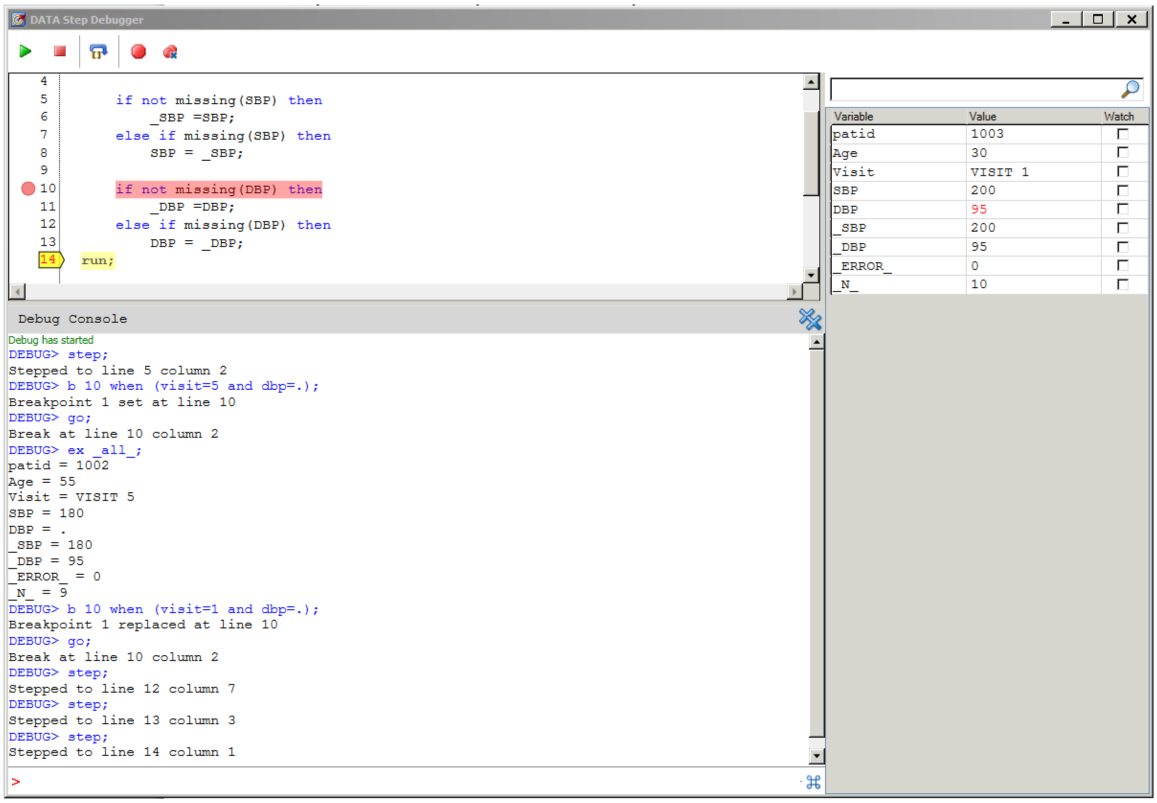


Figure 6. Conditional Breakpoints Set and Results in Watch Window

It seems that the value of DBP at visit 1 for patient 1003 is carried from the last record from patient 1002. If you set the _DBP to missing before assigning _DBP to DBP when reading the first record for each patient, the problem will be solved. You can manually change the variable _DBP to missing in the watch window (see figure 7), or use SET command '`set _dbp=.`' through the command line.

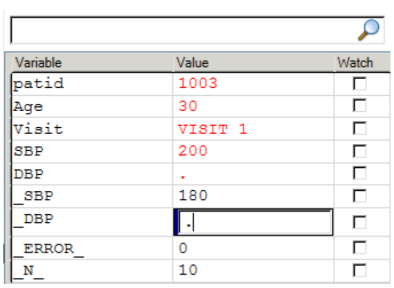


Figure 7. Change values in Watch Window

Based on the deduction above, one statement is added to correct the program by resetting the intermediate variables _SBP and _DBP to missing when executing the first observation of each subject :

```
/***Initialize _SBP _DBP to missing***/
    if visit=1 then
          do;
                _SBP = .;
                _DBP = .;
          end;
```

PROGRAM 2: DEBUGGING MERGE STATEMENT

Generally in clinical trial, there are varies data sets to archive different information of a patient, for example, the vital signs data set to store measurement results of vital signs, the demographics data set to store the demographics information (age, sex, race etc.) of patients.

Suppose there is a data set Demog.sas7bdat (see figure 8), which contains the demographics information (age, sex, weight) and drug treatment group information (ARMCD) for patient 1001, 1002 and 1003. To evaluate the treatment effect, an analysis data set needs to be created by combining the demographics and vital signs data together.

	patid	Age	Sex	ARMCD	WEIGHT
1	1001	23	F	A	185
2	1002	55	M	A	170
3	1003	30	F	B	160

Figure 8. Demographics Data Demog.sas7bdat

The following program is used to merge the Demog.sas7bdat and vitals_1.sas7bdat by matching patient number. Meanwhile, we want to convert the unit of WEIGHT from pounds to kilograms.

```
proc sort data=demog;
    by patid;
run;

proc sort data=vitals_1;
    by patid;
run;

data vitals_2;
    merge demog vitals_1;
    by patid;

    /*change weight from Pounds to kilograms*/
    weight = weight / 2.2;
run;
```

Again, this program is executed successfully without any error message displayed in the log window. The resulting data set is showed in figure 9, which looks quite weird. Notice that the value of WEIGHT changes for each record, even within the same patient.

	patid	Age	Sex	ARMCD	WEIGHT	Visit	SBP	DBP
1	1001	23	F	A	84.090909091	VISIT 1	120	80
2	1001	23	F	A	38.223140496	VISIT 2	119	80
3	1001	23	F	A	17.374154771	VISIT 3	120	78
4	1001	23	F	A	7.8973430777	VISIT 4	120	82
5	1001	23	F	A	3.5897013989	VISIT 5	120	80
6	1002	55	M	A	77.272727273	VISIT 1	188	96
7	1002	55	M	A	35.123966942	VISIT 2	188	90
8	1002	55	M	A	15.965439519	VISIT 4	182	95
9	1002	55	M	A	7.2570179633	VISIT 5	180	95
10	1003	30	F	B	72.727272727	VISIT 1	200	.
11	1003	30	F	B	33.05785124	VISIT 2	199	99
12	1003	30	F	B	15.026296018	VISIT 3	190	99
13	1003	30	F	B	6.8301345537	VISIT 4	189	90
14	1003	30	F	B	3.1046066153	VISIT 5	190	98

Figure 9. Resulting Data Set

To figure out the issue, launch the DATA Step Debugger to investigate what's going on in the PDV. You can use the CALCULATE command to double check whether the calculation is correct or not. And use WATCH command to track any changes made to the WEIGHT variable. Setting a watch on WEIGHT variable suspends program execution whenever the variable value is changed. To set watch on the WEIGHT variable, locate it in the watch window and click the check box. Then click STEP icon on the toolbar to continue execution until the value of WEIGHT changes. Notice that the automatic variables such as _N_ and _ERROR_ are included in the watch window. In this case the FIRST.patid and LAST.patid are included as well because there's BY-group processing in the program. See figure 10.

Stepping through, you can see the value of WEIGHT was calculated based on the imputed value instead of original value. To solve this problem, rename WEIGHT to WEIGHT_LBS to keep the original value, and derive WEIGHT by using the renamed variable:

```
merge demog(rename=(weight=weight_lbs)) vitals_1;
    by patid;

  /*change weight from Pounds to kilograms*/
    weight = weight_lbs / 2.2;
```

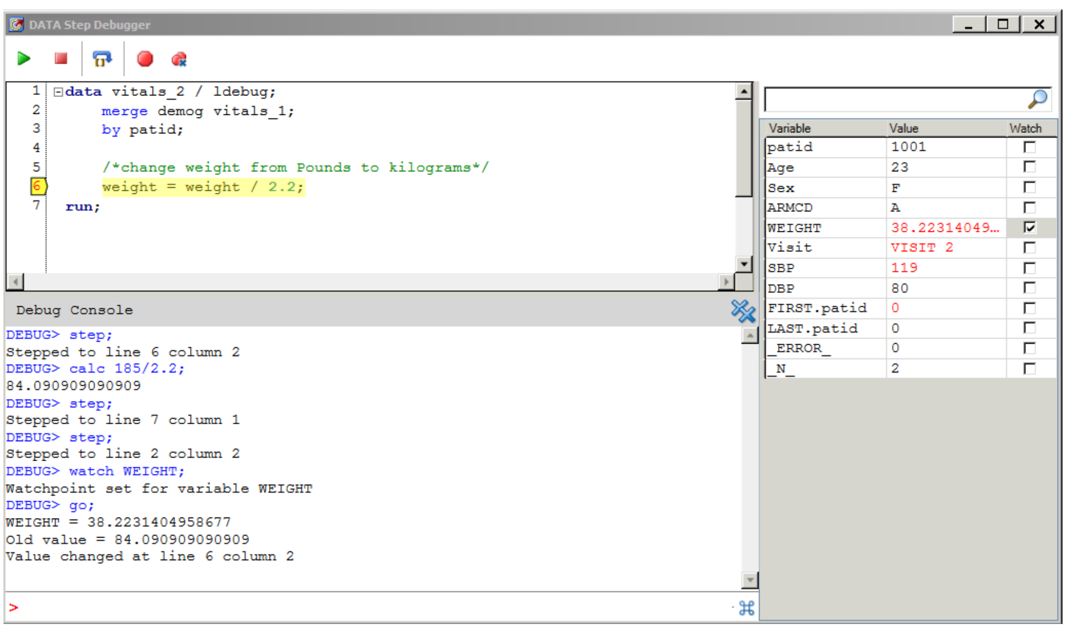


Figure 10. Debugging Window for MERGE Statement

PROGRAM 3: DEBUGGING DO LOOPS

Let's consider the data set generated in program 2, for each patient, the SBP is measured at different visits. Now we want to get an average value of SBP for each patient considering all visits.

The following program used DO loops to calculate the average value:

```
proc sort data= vitals_2;
   by patid;
run;

data vitals_3 (drop=n total);
   do n = 1 by 1 until (last.patid);
        set vitals_2;
        by patid;
        total = sum(SBP);
   end;

   mean_sbp = total/n;

   do until (last.patid);
        set vitals_2;
        by patid;
        output;
   end;
run;
```

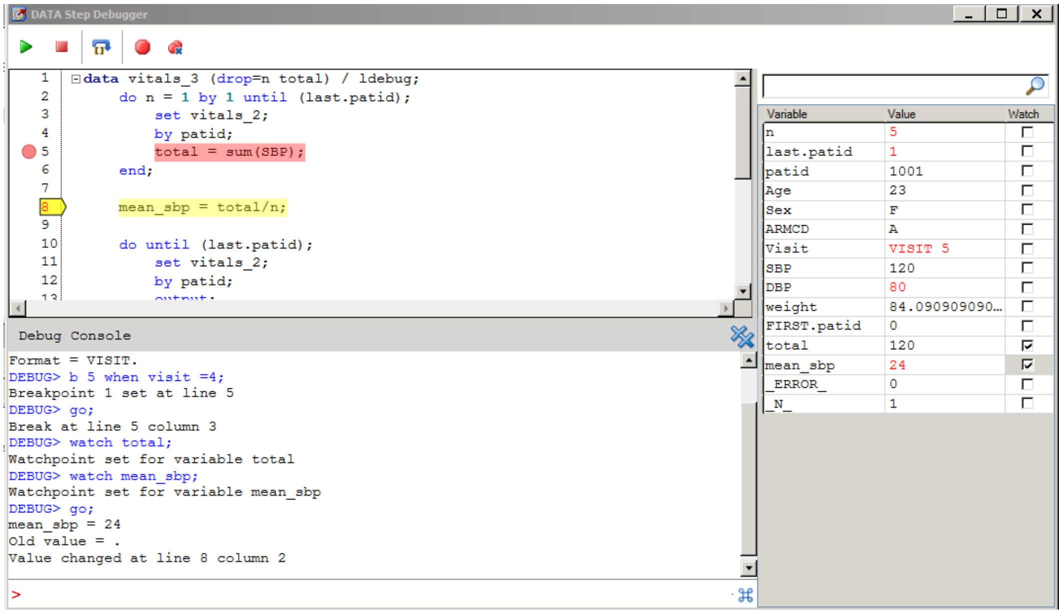
Here is the resulting data set, unfortunately, the average value in variable mean_sbp seems not correct.

	patid	Age	Sex	ARMCD	Visit	SBP	DBP	weight	mean_sbp
1	1001	23	F	A	VISIT 1	120	80	84.090909091	24
2	1001	23	F	A	VISIT 2	119	80	84.090909091	24
3	1001	23	F	A	VISIT 3	120	78	84.090909091	24
4	1001	23	F	A	VISIT 4	120	82	84.090909091	24
5	1001	23	F	A	VISIT 5	120	80	84.090909091	24
6	1002	55	M	A	VISIT 1	188	96	77.272727273	45
7	1002	55	M	A	VISIT 2	188	90	77.272727273	45
8	1002	55	M	A	VISIT 4	182	95	77.272727273	45
9	1002	55	M	A	VISIT 5	180	95	77.272727273	45
10	1003	30	F	B	VISIT 1	200	.	72.727272727	38
11	1003	30	F	B	VISIT 2	199	99	72.727272727	38
12	1003	30	F	B	VISIT 3	190	99	72.727272727	38
13	1003	30	F	B	VISIT 4	189	90	72.727272727	38
14	1003	30	F	B	VISIT 5	190	98	72.727272727	38

Figure 11. Vitals_3.sas7bat

In the debugger window, set a breakpoint at line 5 at the 4th iteration, execute the program and see the changes in the watch window. Click the check box to monitor the changes of variable total and mean_sbp.

Step through the code, you'll find that the variable total was set to the last record, instead of the sum of all records at different visits for each patient.



Update the code as below to get the cumulative summary:

```
total = sum(total, SBP); /***cumulative summary***/
```

The three examples demonstrated how to debug DATA step logic errors in different situations. Note that the DATA step debugger does not support debugging datasets which contain DATALINES or CARDS statements. To use this feature please store your data in an external file.

CONCLUSION

The logic errors in DATA Step are common and difficult to identify and correct. The DATA Step Debugger in SAS Enterprise Guide provides programmers a user-friendly interface to dig into your DATA step code and detect the logic errors. As this paper demonstrated, the debugger is a powerful tool to control the execution of DATA step processing.

REFERENCES

SAS Institute Inc. 2013. "DATA Step Debugger" chapters in *Base SAS® 9.4 Utilities: Reference.* Cary, NC: SAS Institute Inc. Available at http://support.sas.com/documentation/cdl/en/lebaseutilref/64791/PDF/default/lebaseutilref.pdf.

Hemedinger, Chris, Using the DATA step debugger in SAS Enterprise Guide, The

SAS Dummy Blog, http://blogs.sas.com/content/sasdummy/2016/11/30/data-stepdebugger-

CONTACT INFORMATION

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

Name: Mina Chen
Enterprise: Roche (China) Holding Ltd.
Address: 4F, Building 11, 1100 Longdong Avenue, Pudong New District
City, State ZIP: Shanghai, China
Work Phone: +86-021-2892 3475, 201203
E-mail: mina.chen@roche.com