PharmaSUG 2012 – Paper PO17

Basic Debugging Techniques

Beatriz Garcia, Pharmanet/i3, Mexico City
Alberto Hernandez, Pharmanet/i3, Mexico City

ABSTRACT

When we are working with a large amount of data, tracking down logic errors in our code may be difficult to do; for example, the size of a Data Set may make it difficult to print, and manually tracing through record processing can take a long time.

Since debugging is the process of removing logic errors from a program, this paper describes some useful techniques like using a simple %PUT statement; PUT statement or the DATA Step Debugger(SAS® 9.2) to facilitate and speed up the debugging of your SAS code.

INTRODUCTION

Debugging is the process of removing logic errors from a program. Unlike syntax errors, logic errors do not stop a program from running, instead, they cause the program to produce unexpected results.

When the code is not working as expected, the starting point is to discover what it is doing.

This paper will present some useful techniques for all programmers, from beginners to advanced.

- %PUT Statement
- PUT Statement
- DATA Step Debugger

%PUT STATEMENT

Adding a % before the PUT statement displays the values of macro variables in the log.

In this case, is needed to know how many patients are for treatment; so it is useful for tracking the total of population.

Example:

```
DATA TTreatment;
  INPUT Patient $ Rtrtn;
  DATALINES;
  1001     1
  1002     2
  1003     1
  1004     1
  1005     2
  1006     1
  1007     1
  1008     2
  1009     2
  1010     2
  ;
Run;

Proc sql  noprint;
  Select count(distinct PATIENT) into:RND1 from TTreatment where RTRTN = 1;
  Select count(distinct PATIENT) into:RND2 from TTreatment where RTRTN = 2;
  Select count(distinct PATIENT) into:RND3 from TTreatment where RTRTN in (1,2);
Quit;
```

```
%put &rnd1 &rnd2 &rnd3;
```

LOG Output

```
222  %put &rnd1 &rnd2 &rnd3;
5        5       10
```

PUT STATEMENT

CASE 1

PUT Statement allows the SAS programmer to evaluate whether a SAS function is working properly

Example:

```
DATA _NULL_;
  Weight=65;
  Height=1.51;
  BMI= Weight/(Height*Height);
  Put BMI;
Run;

LOG Output
28.5075216
```

Using a PUT statement after BMI evaluation, SAS LOG will indicate how data values are being transformed.

CASE 2

PUT Statement allows the SAS programmer to create messages that indicate something wrong is happening, it can generate an alert message indicating which observation is failing.

Example:

```
DATA Temp1;
  INPUT id $ Weight Height;
  DATALINES;
  001      65     1.52
  002             53     1.52
  003             .      1.65
  004             66     .
  005             .      .
  ;
Run;

DATA Temp2;
  Set Temp1;
  If weight=. Then PUT "Weight is missing for ID: " ID=;
Run;
```

```
The following is written to the SAS Log:
67   DATA Temp1;
68   INPUT id $ Height Weight;
69   DATALINES;

NOTE: The data set WORK.TEMP1 has 5 observations and 3 variables.
NOTE: DATA statement used (Total process time):
      real time           0.04 seconds
      cpu time            0.00 seconds


75   ;
76   Run;
77
78   DATA Temp2;
79       Set Temp1;
80       If weight=. Then PUT "Weight is missing for ID: " ID=;
81   Run;

Weight is missing for ID: id=004
Weight is missing for ID: id=005
NOTE: There were 5 observations read from the data set WORK.TEMP1.
NOTE: The data set WORK.TEMP2 has 5 observations and 3 variables.
NOTE: DATA statement used (Total process time):
      real time           0.06 seconds
      cpu time            0.00 seconds
```

DATA STEP DEBUGGER

The DATA Step Debugger is part of Base SAS software. By issuing commands, you can execute DATA step statements one by one and pause to display the resulting variable values in a window. Then, observing the results that are displayed, you can determine where the logic error lies.

In this example we can detect the logic error easily since we want to show the main function of the DATA Step Debugger. Is recommended to use the DATA Step Debugger when there is a large amount of code or Data Set is too big.

Example:

```
DATA Temp1;
  INPUT id $ Weight Height;
  DATALINES;
  001            65     1.52
  002            53     1.52
  003            .      1.65
  004            66     .
  005            .      .
  ;
Run;

DATA Temp2;
  Set Temp1;
  If height =. then height =0; /*This will cause an unexpected result*/
  BMI= Weight/(Height*Height);
Run;
```

This is the unexpected result.

LOG OUTPUT

```
NOTE: Division by zero detected at line 149 column 17.
id=004 Weight=66 Height=0 BMI=. _ERROR_=1 _N_=4
```

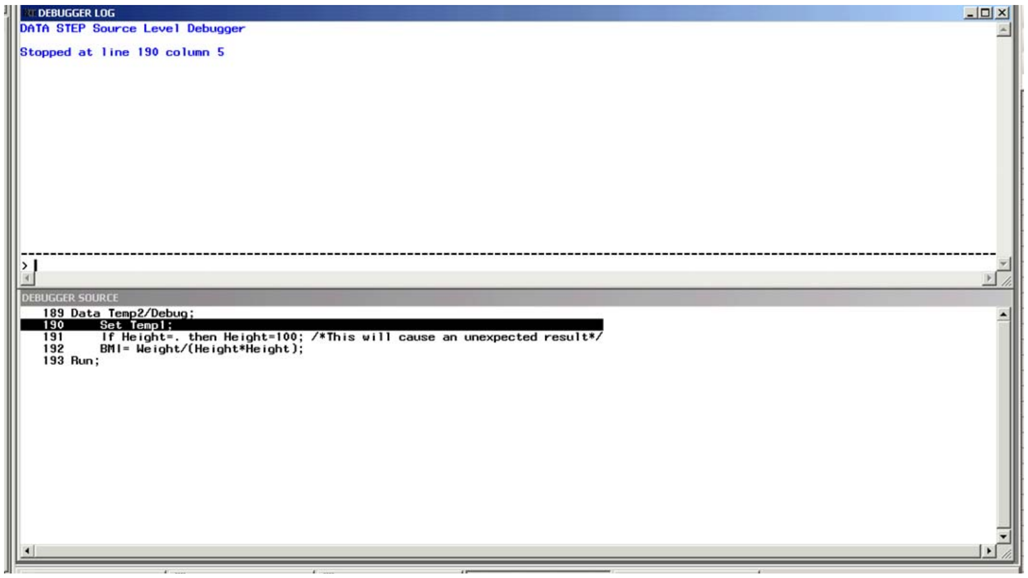
A statement has been added to set the value of HEIGHT to 0 if it is missing. However, this will cause an unexpected result in our output data set. If there are many records in our data set, then DATA Step Debugger becomes useful for locating logic errors.

The DATA Step Debugger is triggered by adding the DEBUG option at the end of the Data statement:.

```
DATA Temp2/Debug;
```

The screen splits into two debugger windows; the upper window is the DEBUGGER LOG window, where you can issue debugger commands by typing them on the debugger command line (marked by a > prompt).

The lower window is the DEBUGGER SOURCE window. DATA step execution pauses just before the execution of the highlighted statement.



```
DATA STEP Source Level Debugger

Stopped at line 190 column 5
```

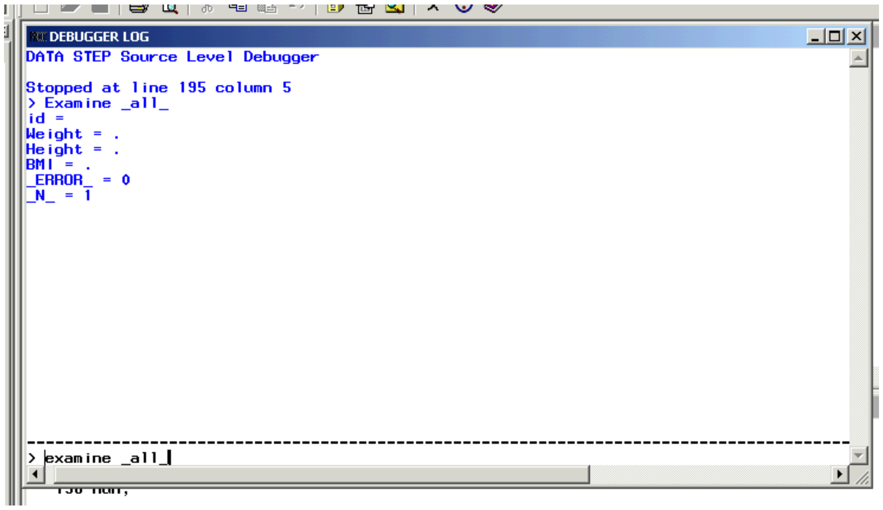
```
>
```

```
189 Data Temp2/Debug;
190     Set Temp1;
191     If Height=. then Height=100; /*This will cause an unexpected result*/
192     BMI= Weight/(Height*Height);
193 Run;
```

Display 1. DATA Step Debugger.

EXAMINE COMMAND.

The EXAMINE command displays the values of one or all variables in the Program Data Vector before execution begins.



```
DATA STEP Source Level Debugger

Stopped at line 195 column 5
> Examine _all_
id =
Weight = .
Height = .
BMI = .
_ERROR_ = 0
_N_ = 1
```

```
> examine _all_
```

Display 2. Examine command.

Its acronym is:

```
e _all_
```

Or you can examine a specific variable:

```
E bmi
```

Check Appendix A for more commands.

Check Appendix B for syntax.

CONCLUSION

The %PUT statement is a very simple instruction, but it is useful when we want to look at totals, numbers or specific data.

The PUT statement is very helpful when you need to evaluate if a SAS function is working properly or you want to display any alert message.

If we want to find out how the code is working, the DATA Step Debugger is a very powerful tool for all levels of programmers. Its interactive environment allows us to look directly at how the data is processed in order to identify why we are not getting the expected results.

We conclude that if we have few code is recommended to use %PUT Statement if you want to look any total or PUT Statement if you need to evaluate if a SAS function is working properly.

If we want to dive into a large amount of code and the logic error is not easy to identify or to know the behavior of the variable, the best suggestion would be the DATA Step Debugger.

RECOMMENDED READING

- Base SAS® Procedures Guide
- SAS® For Dummies®

CONTACT INFORMATION

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

Name: Beatriz Garcia
Enterprise: Pharmanet-i3
Address: Insurgentes Sur #716
City, State ZIP: Mexico City, 03100
Work Phone: +52 55 2210 0181
E-mail: begarcia@pharmanet-i3.com

Name: Alberto Hernandez
Enterprise: Pharmanet-i3
Address: Insurgentes Sur #716
City, State ZIP: Mexico City, 03100
Work Phone: +52 55 5005 5507
E-mail: ahernandez@pharmanet-i3.com

APPENDIX A.- DEBUGGER COMMANDS BY CATEGORY

Category	DATA Step Debugger	Description
Controlling Program Execution	GO	Starts or resumes execution of the DATA step
	JUMP	Restarts execution of a suspended program
	STEP	Executes statements one at a time in the active program
Controlling the Windows	HELP	Displays information about debugger commands
	SWAP	Switches control between the SOURCE window and the LOG window
Manipulating DATA Step Variables	CALCULATE	Evaluates a debugger expression and displays the result
	DESCRIBE	Displays the attributes of one or more variables
	EXAMINE	Displays the value of one or more variables
	SET	Assigns a new value to a specified variable
Manipulating Debugging Requests	BREAK	Suspends program execution at an executable statement
	DELETE	Deletes breakpoints or the watch status of variables in the DATA step
	LIST	Displays all occurrences of the item that is listed in the argument
	TRACE	Controls whether the debugger displays a continuous record of the DATA step execution
	WATCH	Suspends execution when the value of a specified variable changes
Tailoring the Debugger	ENTER	Assigns one or more debugger commands to the ENTER key
Terminating the Debugger	QUIT	Terminates a debugger session

APPENDIX B.- SYNTAX

```
BREAK location <AFTER count> <WHEN expression> <DO group >

CALC expression

DELETE BREAK location
DELETE WATCH variable(s) | _ALL_

DESCRIBE variable(s) | _ALL_

EXAMINE variable-1 <format-1> <...variable-n <format-n>>
EXAMINE _ALL_ <format>

GO <line-number | label>

LIST _ALL_ | BREAK | DATASETS | FILES | INFILES | WATCH

QUIT

SET variable=expression

WATCH variable(s)
```