Paper HW08

An Animated Guide: The Data Step Debugger

Russell Lavery, Contractor, About Consulting, Chadds Ford, PA

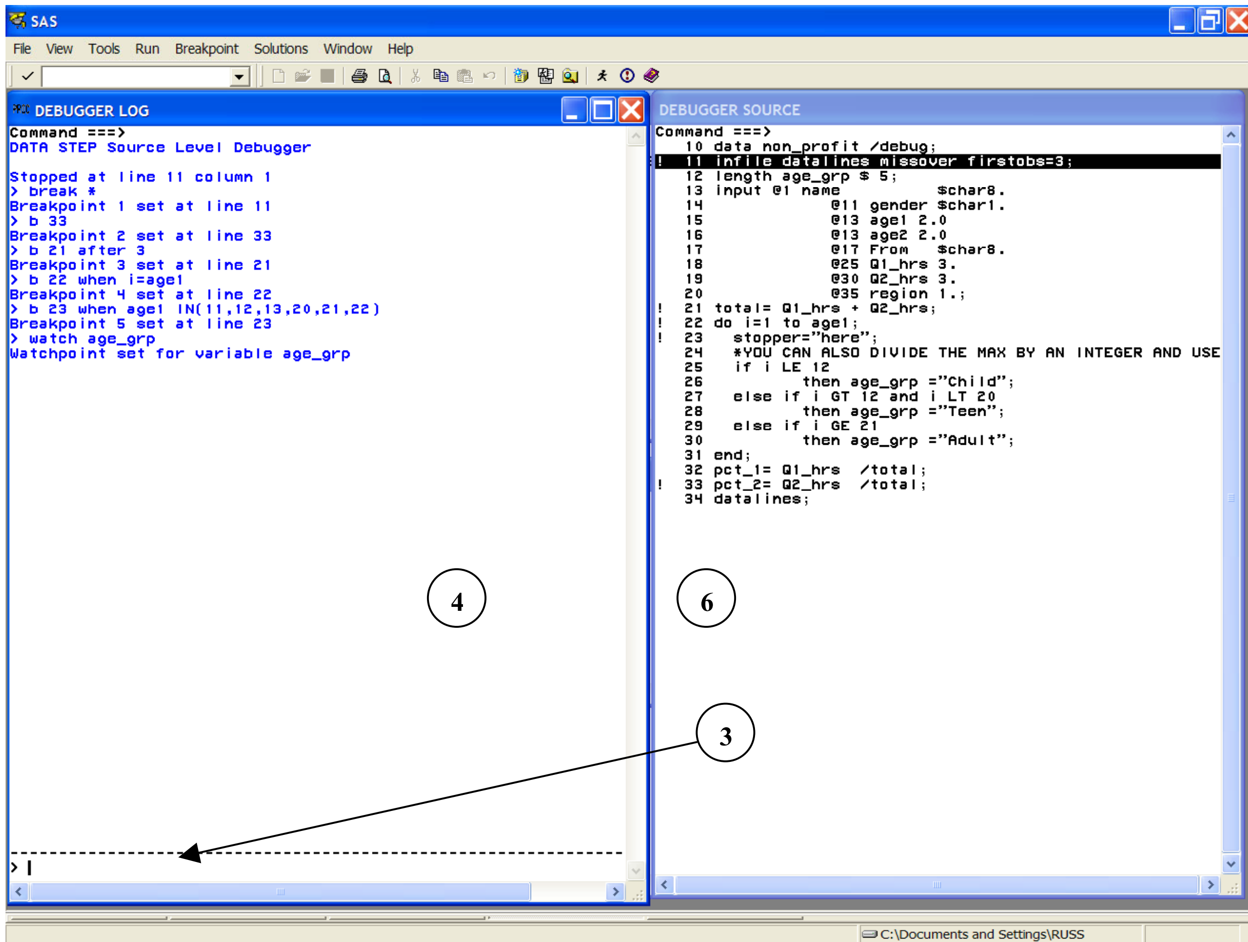

Figure 1

ABSTRACT

Use of the Data Step Debugger (DSD) simplifies the job of debugging code and, while the data step debugger (DSD) was introduced in SAS® version 6.11, it is not widely known. It is a powerful tool gives you control over the execution of "data step" code and allows you to watch your program execute- line by line. The DSD allows you to watch if-else statements execute and check if the code is correct while the code is running, rather than checking for correctness by writing a series of Proc Print statements. The DSD allows you to not only watch code execute, but will allow you to MANUALLY CHANGE the values in the program data vector as your job is running.

The goal of a programmer learning the DSD should be to learn the DSD commands and how to link them to a key. Linking a series of commands to a key makes the DSD much more powerful.

INTRODUCTION

This paper concentrates on using the DSD interactively and on how the DSD can be used to boost productivity. It will quickly cover commands, and then concentrate on explaining the combinations of commands, and the programming situations, where the DSD is likely to give a productivity boost. The goal of a programmer learning the DSD should be to learn the DSD commands and how to link them to a key. Linking a series of commands to a key makes the DSD much more powerful.

USING THE DSD IN INTERACTIVE MODE

The debugger is most often used in interactive mode. To operate in this manner, SAS must be running with the SAS editor window active. The data step you are trying to debug must not have syntax errors, only logical errors. If you leave out a semicolon and run the DSD, DSD screens will not be displayed. Referencing non-existing variables might cause SAS to display the DSD windows but not stop at break points. In short, syntax errors cause the DSD to misbehave.

The DSD is invoked by adding "/debug" to the data step line [circle (1) Figure 1], highlighting the data step code and then hitting <F3>. Invoking the DSD will add two more windows to the screen, the DSD source and the DSD log. Since the debugging work is done in these two windows, programmers typically hide the SAS log, list and editor windows so as to be easily able to see activity in the DSD log and editor.

A workable screen layout is shown in Figure 1. The messages shown in the DSD log are rather short and programmers often make the DSD editor wider. Once you have found an effective layout, you should issue WSAVE on the command line [circle (2) Figure 1] in both windows (note that only the active window shows its command line) and SAS will open the DSD with that layout in future sessions. To make the command line appear work through the following steps: Tools-Options-Preferences-Select the View tab – click on command line and OK. When the command lines appears issue the WSAVE command in each window.

Running the DSD is fairly simple. DSD commands are typed in a command line in the bottom of the log window (circle (3) Figure 1). The line about to execute is highlighted in black in the DSD editor [circle (4) Figure 1]. The DSD only stops on (reverse highlights) executable lines. In the code in Figure 1, the DSD will never stop with line 12 in reverse highlight because line 12 is not executable. Because the DSD only stops on executable lines, it is sometimes useful to insert executable "nonsense lines" like:

```
Book="mark1";
```

into the code so that you will have an executable line where you'd like the DSD to stop. Progress messages [circle (5) Figure 1], as well as the results of DSD queries are written to the DSD log.

DSD COMMANDS

There are only a handful of easy-to-learn DSD commands. However; using sequences of commands, not individual commands, is what makes the DSD a useful tool. ***Binding a useful, and frequently used, series of commands to a key-combination is the starting point for DSD power use. It should be a goal of everyone starting to learn the DSD.*** The DSD is painfully slow if you if you are a bad typist and have to type in every command.

DSD COMMAND GROUPS

The DSD commands are generally grouped by the functions they perform and are grouped that way below. The commands have abbreviations to reduce typing and options to increase their power/flexibility. The most frequently used commands have been underlined. Many commands have abbreviations that can be discovered in the SAS help pages. Generally, binding commands to a key minimizes the need to use the abbreviations.

Execution commands move the "active line" (the reverse highlighted line) of the SAS code- by executing steps or skipping steps.

<u>Step</u> **<u>Go</u>** **<u>Jump</u>** **<u>Quit</u>**

Suspension/Break commands specify where to stop execution of your SAS code

<u>Break</u> **<u>Watch</u>** **<u>Delete</u>** **Trace**

Display commands show the PDV and input buffer in the DSD LOG.

<u>Examine</u> **List** **Describe**

Window command toggle the active window between editor/log

Swap **mouse-click**

Other commands do miscellaneous things

Calculate **<u>Set</u>** **Help**

SPEED TRICKS make the DSD convenient to use

<u>Binding Commands to Keys</u> **<u><F4></u>** **Do-Loop** **Macro**

DSD EXECUTION COMMANDS

Execution commands cause lines of SAS code to execute or cause execution to shift to over lines without executing them. These commands do not print any information to the DSD log.

Step causes the DSD to execute lines. Return is abbreviated <ret> in this paper. *Step<ret>* (or *st<ret>* or *step 1<ret>* or *<ret>*) will cause the editor to execute one command and stop. *Step n* will cause the editor to execute n commands. Usually, if you want to execute one, or a few lines, you will hit the return key, rather than typing *step*. If you want to execute a number of lines, the *Go linenumber<ret>* command can be easier than *step n,* when n is a large number.

Go has several options and does several things. A simple Go <ret> will execute lines until the DSD reaches a stop point. The stop point might be a line on which you have set as a stop point [circle (6) Figure 1] or the “end of the data step”. If you have not set any stop/break points, go will cause processing of all observations through all the lines in the data step. It will run the data step “to a normal conclusion”. If you have previously set a break point at a line, or instructed the DSD to watch a variable for changes in value, issuing *Go<ret>* will cause the DSD to execute lines until the break line is reached or the watched variable changes its value.

Two useful Go options are:

***Go** linenumber <ret>*. This command will cause the DSD to execute all lines between the current line and the specified line number. Line numbers are shown in the left-hand side of the Data Step Source window. You will likely use *Go linenumber <ret>* rather than *step many-n<ret>*.

***Go** label<ret>.* You can put a SAS label in your code and issue the command *Go label<ret>*.

Jump is a command that lets you skip lines of code. If your DSD window showed the screens in Figure 1, and you issued jump 91<ret>, you would not execute any of the lined of code between lines 88 and 91. The jump command can be useful if you think one block of code is causing a problem, and you do not want to have it run but do not want to take the effort to comment if out.

Quit ends the debugger session. It closes the debugger log and debugger editor.

DSD SUSPENSION/BREAK COMMANDS

Suspension/break commands specify where/when to stop execution. You can tell the DSD to stop on a line, or stop on a line when a condition is true. You can set several break points, each having different logical conditions. You can also ask the DSD to stop immediately, if a variable changes value. When you issue a "suspension command you will get a message in the DSD log and see an ! in the DSD editor (see line 88 Figure 1) on the “break line”. You usually break, or suspend execution, so that you can then issue “display commands” to check values of variables as execution waits on that line. Printing commands are separate.

Break tells the DSD to "stop on a line" and has useful options.

Sometimes you will want to stop *every* time the DSD is about to execute the line. If the DSD is highlighting the line you wish to make a "break every time line" you can type break *<ret> [see Debugger log, Figure 1] and the line will become a "break every time you are about to execute" line. Generally, If you want to make a line a "break every time you are about to execute " line, you issue the command *break linenumber<ret>*. When a break has been set successfully, the DSD writes a message to the DSD log and puts an exclamation point on that line in the DSD Source Window [circle (6) figure 1].

When your programming problem is to try to find "odd observations", or you are trying to debug a do loop, "break at this line when" is a useful option. Often you want to see the last “pass through a loop” and you would like to see the processing of observations through the nesting of the if-else if code. Detailed debugging requires that we check the execution of every if statement. The DSD will stop execution of code at the break line only when the condition is true. The "Break at this line when" is useful for debugging loops and for examining what happens when certain observations pass through the data step [see the break commands in Figure 1].

You might want to check that each level in an if statement executes correctly (See the If series in Figure 1 for background). We might like to check the proper routing of observations that are close to the critical values of the if statements (11, 12, 20, 21). We see that observations with age equal eleven should be classified as a child. Observations with ages twelve through twenty should go to teen. Observations with age equal Twenty-one should be classified as adult.

A useful breakpoint for the example above might be *Break 91 when age in (11,12,13,19,20,21)<ret>* followed by a *GO*. This would process observations, possibly very many observations, and stop processing on line 91 when an observation with an age close to the "if breakpoints" is about to be processed. After breaking execution on an interesting observation you could then proceed to "step" that observation through the data step – one line at a time.

The debugger log, in Figure 1, also shows the issuance of a "break line-number after n" command. This is also useful in debugging loops. You can tell the DSD to stop processing after a line has been passed n times. This means you can instruct the DSD to cycle through a do loop and then stop when it is just about to leave the loop. You can then step through the last cycle and see what happens as you leave the loop. [see Figure 1, especially “break 23 when i=age1”].

Watch tells the DSD to stop when a variable changes its value, *regardless of what line is executing*. The command is *watch var.-name<ret>* and does not take any arguments (Figure 1). The watch command is especially useful when you are trying to debug the setting of a flag, especially when the flag can be set in several places in your code.

Delete is maintenance command that “clears” break and watch points. Sometimes, you can use the DSD to examine two, or three, problems. If you resolve the first problem you will want to clear the break points and the watch variables associated with that first problem. You might then set new breaks/watch variables to investigate the second problem.

Delete will clear both breakpoints and watchpoints. The command to clear a break is *Delete B linenumber<ret>*. The command to clear a watchpoint is *Delete W varname<ret>.* You can save time with *D B _all_<ret>* or *D W _all_*<ret>.

Trace can be toggled on and off. When trace is on, messages are written to the log that show the lines that were executed. This can be useful if you issue a go and want to be able to determine what path execution took through an if-else section. Its use is limited to tracking execution through line numbers and does not provide any information about variables changing values.

DSD DISPLAY COMMANDS

Display commands show values, or attributes, of variables. You set break points so that you can issue display commands at interesting points in your program. Once you have halted execution with a break you will use display commands to show the PDV and input buffer.

Examine is the most useful display command. It displays values from the PDV in the DSD log. The syntax is *examine variable(s) format<ret> .* Examine <varnames> format will show, in the DSD log, current PDV values for the variables listed. The format is not a required option but can be very useful in making output more readable. It applies a format to the value in the PDV.

To save keystrokes you can issue the command as *E _all_<ret>*. This command will examine (show in the DSD log) all variables in the PDV. While this is an “easy to type” command it usually provides more output than is desired, or convenient. Repeatedly issuing this command with many variables (e.g. examine var1 var12 var13 var24 var5) can be tedious. To save typing, the examine command is usually bound to a function key (explained below) or the programmer uses <F4>.

Examine does not accept the following SAS abbreviations:
Examine _numeric_ ***or*** Examine _character_ ***or*** e var_nm1-var_nm3 ***or*** e varnm1--varnmchar

List is a DSD display tool and writes messages to the DSD log to describe your DSD session. It has several options
List d<ret> provides information about the SAS data set you are creating. *List I<ret>* will list the current infile (the source of data) and display the input buffer. Imagine, you are reading a text file and have made a mistake in coding the input statement (maybe you have typed in the wrong column number). You can use *list I<ret>* to see the input buffer and then use *examine varname<ret>* to see if what you read into the PDV is what SAS has in the input buffer. The input buffer shows what is in the data file and the PDV shows what you "got" into your variables.

List b<ret> writes a message that tells you what lines have breakpoints set, but it produces very sparse messages. Remember that we can associate conditions with the breakpoints when we set breakpoints. We might set breakpoints with associated conditions like "when var=" and "after n". However, *list b<ret>* will not report on the conditions that are associated with the breakpoints. It just reports the line number, not the associated condition.

List W<ret> will write a message to the DSD log that indicates the variables you are watching and tells the current values of the variables.

List _all_<ret> will report on all the list options.

Describe writes information about variables to the DSD log. The format is *Describe firstvar secvar thirdvar<ret>.* Describe causes SAS to write the variable name, type and length to the DSD log.

DSD WINDOW COMMANDS

The window command toggles the active window between editor and log. This is useful because only the active window has a scroll bar and a command line. You will issue the WSAVE command on the command line of every window whose shape you want SAS to remember. To scroll a window, it must be active. If your data step does not fit on one screen, you might want to scroll up/down to see code that is not currently displayed.

You can change the active window with *Swap<ret>* or by left mouse clicking on the window you wish to make active.

DSD SPEED TRICKS

Using speed tricks makes the DSD a powerful tool and a pleasure to use. The most important speed trick is binding a set of commands to a key (or pair of keys) on your keyboard.

Figure 2

Figure 3

If the DSD is active and you press <F9> the DSD keys window will be displayed [see Figure 2].

Figure 2 shows that two types of commands have been linked to keys: command line commands (like help, reshow, end, recall, etc) and DSD "command series" (see shift <F1> and shift<f11>). Note that <F9> has already been linked to the keys command. That is why pressing <F9> displays the DSD keys window.

The keys window shows the key bindings that are in effect when the author's DSD is running. These key bindings are not valid when regular SAS is running. If I were to press <F9>, when in the regular SAS editor, a keys window would be displayed. However, the key bindings would be different from the DSD key bindings shown above. Binding commands to keys is a critical productivity technique for using the DSD.

Looking in Figure 2, shift <F11> shows a variation of one of the most useful DSD command strings. Pressing Shift<F1> will execute one line and then display the full PDV. If there were repeated need to examine four or five variables you could save time by binding the examine command (e.g. DSD Step; DSD examine var1 varB var 23 varAA;) to a key and you would never need to type the command again.

Key binding is easy to do. Just open the key window and use the mouse to position the cursor on a line. Type in the DSD commands remembering just two rules. 1) DSD commands must begin with the string DSD and 2) DSD commands must end with a semicolon. Several commands can be linked to one keystroke. (You can see examples of these rules in Figure 2. – see shift-<F1> and shift-<F11>) To finish the key binding process, close the keys window to have a better view of events. Finally, make the DSD log window active. Can then press the key(s) to which you have bound the commands and watch the DSD execute them.

<F4> is usually bound to the recall command. It has the same function as DOSKEY does in DOS. Pressing <F4> will recall the last twenty DSD commands back to the DSD log command line. Once the proper command is on the DSD log command line, it can be executed. This can be a small timesaver, if you want to re-use a command but do not want to bind the command string to a key.

Multiple DSD commands can be "set to execute" after a break by enclosing them in a Do-End. Often this feature can be mimicked by binding the desired commands to several keys and pressing the keys sequentially. The DSD can also call

macros when conditions are met. The author has not found these either of these DSD features to be very helpful.

Figure 3 (above) shows how the DSD can be run from pull down menus. Right clicking on the DSD editor or DSD log will produce a menu of DSD commands. DSD commands (examine, list, break, delete etc) can be executed from this menu. A very useful feature of the menu is the ability to save the DSD Log, and/or DSD Editor, to a file. From the menu shown in Figure 3, select File - Save As. This "easy save" is the most useful feature on the menu. For most DSD commands, typing commands is usually faster than using a mouse and menu.

ENDING THE DEBUGGER SESSION

The debugger session is ended by typing quit<ret> in the DSD command line of the DSD log.

CONCLUSION

The DSD is a powerful tool for debugging the Data Step parts of programs. It should be in the toolkit of every SAS programmer.

REFERENCES

SUGI 23 paper 25 How to use The Data Step Debugger by S. David Riba (available online).

Conference proceedings are a valuable source of information of SAS topics. Many years of NUGI and NESUG proceedings are on-line and accessible for free. An Excel Spreadsheet listing of the article titles, authors and web locations is available from www. About-Consulting.com.

ACKNOWLEDGMENT

SAS is a registered trademark or trademark of SAS Institute, Inc. in the USA and other countries. ® indicates USA registration.
Other brand and product names are registered trademarks of their respective companies.

CONTACT INFORMATION

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

Russell Lavery- Independent Contractor for About Consulting.com
9 Station Ave. Apt 1, Ardmore, PA 19003, 610-645-0735 # 3
Email: russ.lavery@verizon.net