**Examples**

|  |
| --- |
|  |
| **Example 1: Debugging a Simple DATA Step** |

This example shows how to debug a DATA step when output is missing.

**Discovering a Problem**

This program creates information about a travel tour group. The data files contain two types of records. One type contains the tour code, and the other type contains customer information. The program creates a report listing tour number, name, age, and sex for each customer.

/\* first execution \*/

data tours (drop=type);

input @1 type $ @;

if type='H' then do;

input @3 Tour $20.;

return;

end;

else if type='P' then do;

input @3 Name $10. Age 2. +1 Sex $1.;

output;

end;

datalines;

H Tour 101

P Mary E 21 F

P George S 45 M

P Susan K 3 F

H Tour 102

P Adelle S 79 M

P Walter P 55 M

P Fran I 63 F

;

proc print data=tours;

title 'Tour List';

run;

Tour List 1

Obs Tour Name Age Sex

1 Mary E 21 F

2 George S 45 M

3 Susan K 3 F

4 Adelle S 79 M

5 Walter P 55 M

6 Fran I 63 F

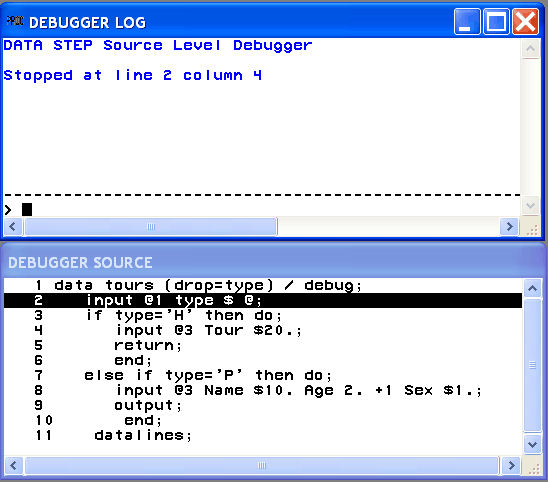
The program executes without error, but the output is unexpected. The output does not contain values for the variable Tour. Viewing the SAS log will not help you debug the program because the data are valid and no errors appear in the log. To help identify the logic error, run the DATA step again using the DATA step debugger.

**Using the DEBUG Option**

To invoke the DATA step debugger, add the DEBUG option to the DATA statement and resubmit the DATA step:

data tours (drop=type) / debug;

The following display shows the resulting two debugger windows.



The upper window is the DEBUGGER LOG window. Issue debugger commands in this window by typing commands on the debugger command line (the bottom line, marked by a >). The debugger displays the command and results in the upper part of the window.

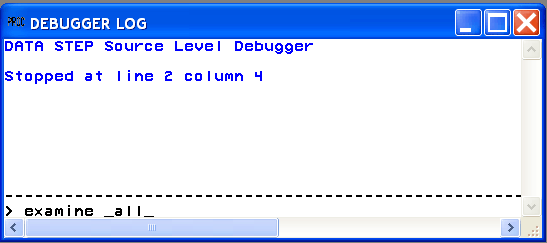
The lower window is the DEBUGGER SOURCE window. It displays the DATA step submitted with the DEBUG option. Each line in the DATA step is numbered with the same line number used in the SAS log. One line appears in reverse video (or other highlighting, depending on your monitor). DATA step execution pauses **just before** the execution of the highlighted statement.

At the beginning of your debugging session, the first executable line after the DATA statement is highlighted. This means that SAS has compiled the step and will begin to execute the step at the top of the DATA step loop.

**Examining Data Values after the First Iteration**

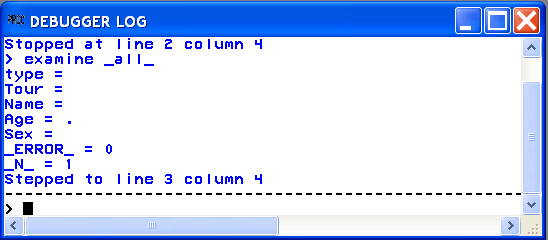
To debug a DATA step, create a hypothesis about the logic error and test it by examining the values of variables at various points in the program. For example, issue the EXAMINE command from the debugger command line to display the values of all variables in the program data vector before execution begins:

examine \_all\_



**Note:**   Most debugger commands have abbreviations, and you can assign commands to function keys. The examples in this section, however, show the full command name to help you find the commands in [Debugger Commands by Category](http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/a000379386.htm#a000379387).  [cautionend]

When you press ENTER, the following display appears:



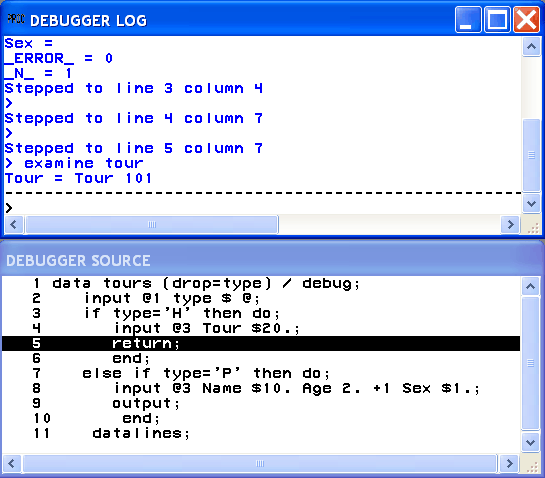
The values of all variables appear in the DEBUGGER LOG window. SAS has compiled, but not yet executed, the INPUT statement.

Use the STEP command to execute the DATA step statements one at a time. By default, the STEP command is assigned to the ENTER key. Press ENTER repeatedly to step through the first iteration of the DATA step, and stop when the RETURN statement in the program is highlighted in the DEBUGGER SOURCE window.

Because Tour information was missing in the program output, enter the EXAMINE command to view the value of the variable Tour for the first iteration of the DATA step.

examine tour

The following display shows the results:



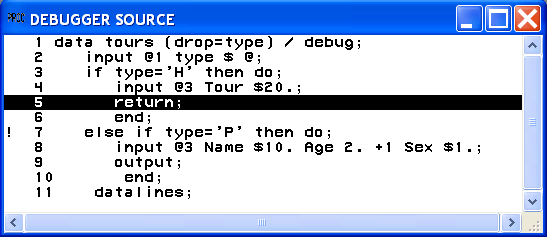
The variable Tour contains the value Tour 101, showing you that Tour was read. The first iteration of the DATA step worked as intended. Press ENTER to reach the top of the DATA step.

**Examining Data Values after the Second Iteration**

You can use the BREAK command (also known as **setting a breakpoint**) to suspend DATA step execution at a particular line you designate. In this example, suspend execution before executing the ELSE statement by setting a breakpoint at line 7.

break 7

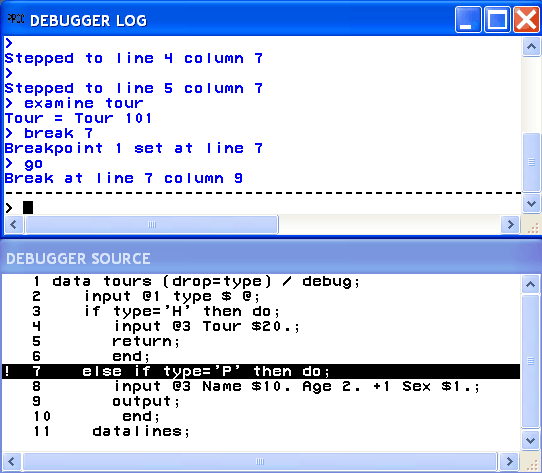
When you press ENTER, an exclamation point appears at line 7 in the DEBUGGER SOURCE window to mark the breakpoint:



Execute the GO command to continue DATA step execution until it reaches the breakpoint (in this case, line 7):

go

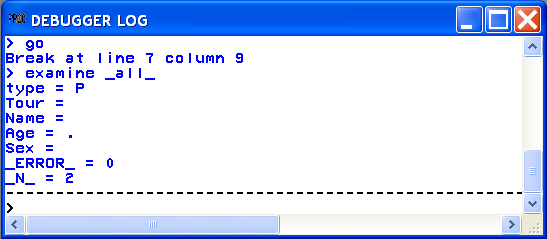
The following display shows the result:



SAS suspended execution **just before** the ELSE statement in line 7. Examine the values of all the variables to see their status at this point.

examine \_all\_

The following display shows the values:



You expect to see a value for Tour, but it does not appear. The program data vector gets reset to missing values at the beginning of each iteration and therefore does not retain the value of Tour. To solve the logic problem, you need to include a RETAIN statement in the SAS program.

**Ending the Debugger**

To end the debugging session, issue the QUIT command on the debugger command line:

quit

The debugging windows disappear, and the original SAS session resumes.

**Correcting the DATA Step**

Correct the original program by adding the RETAIN statement. Delete the DEBUG option from the DATA step, and resubmit the program:

/\* corrected version \*/

data tours (drop=type);

retain Tour;

input @1 type $ @;

if type='H' then do;

input @3 Tour $20.;

return;

end;

else if type='P' then do;

input @3 Name $10. Age 2. +1 Sex $1.;

output;

end;

datalines;

H Tour 101

P Mary E 21 F

P George S 45 M

P Susan K 3 F

H Tour 102

P Adelle S 79 M

P Walter P 55 M

P Fran I 63 F

;

run;

proc print;

title 'Tour List';

run;

The values for Tour now appear in the output:

Tour List 1

Obs Tour Name Age Sex

1 Tour 101 Mary E 21 F

2 Tour 101 George S 45 M

3 Tour 101 Susan K 3 F

4 Tour 102 Adelle S 79 M

5 Tour 102 Walter P 55 M

6 Tour 102 Fran I 63 F

|  |
| --- |
|  |
| **Example 2: Working with Formats** |

This example shows how to debug a program when you use format statements to format dates. The following program creates a report that lists travel tour dates for specific countries.

options yearcutoff=1920;

data tours;

length Country $ 10;

input Country $10. Start : mmddyy. End : mmddyy.;

Duration=end-start;

datalines;

Italy 033000 041300

Brazil 021900 022800

Japan 052200 061500

Venezuela 110300 11800

Australia 122100 011501

;

proc print data=tours;

format start end date9.;

title 'Tour Duration';

run;

Tour Duration 1

Obs Country Start End Duration

1 Italy 30MAR2000 13APR2000 14

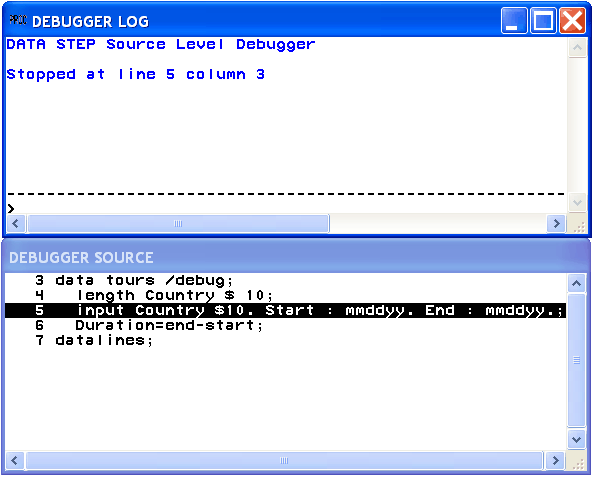
2 Brazil 19FEB2000 28FEB2000 9

3 Japan 22MAY2000 15JUN2000 24

4 Venezuela 03NOV2000 18JAN2000 -290

5 Australia 21DEC2000 15JAN2001 25

The value of Duration for the tour to Venezuela shows a negative number, -290 days. To help identify the error, run the DATA step again using the DATA step debugger. SAS displays the following debugger windows:



At the DEBUGGER LOG command line, issue the EXAMINE command to display the values of all variables in the program data vector before execution begins:

examine \_all\_

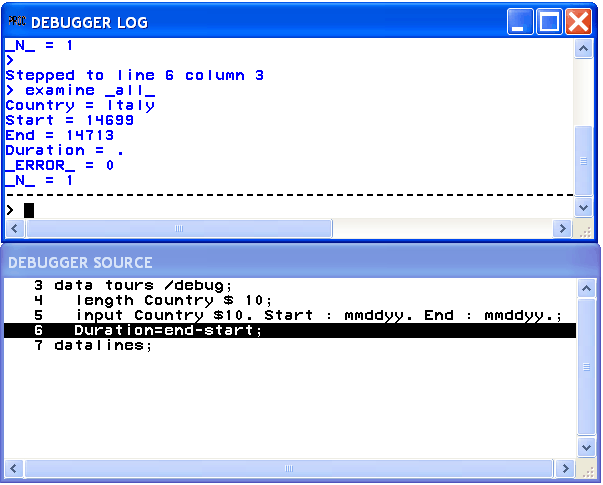
Initial values of all variables appear in the DEBUGGER LOG window. SAS has not yet executed the INPUT statement.

Press ENTER to issue the STEP command. SAS executes the INPUT statement, and the assignment statement is now highlighted.

Issue the EXAMINE command to display the current value of all variables:

examine \_all\_

The following display shows the results:



Because a problem exists with the Venezuela tour, suspend execution before the assignment statement when the value of Country equals Venezuela. Set a breakpoint to do this:

break 6 when country='Venezuela'

Execute the GO command to resume program execution:

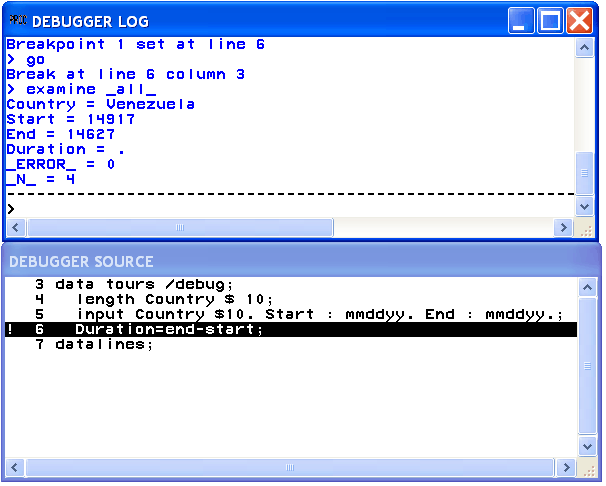
go

SAS stops execution when the country name is Venezuela. You can examine Start and End tour dates for the Venezuela trip. Because the assignment statement is highlighted (indicating that SAS has not yet executed that statement), there will be no value for Duration.

Execute the EXAMINE command to view the value of the variables after execution:

examine \_all\_

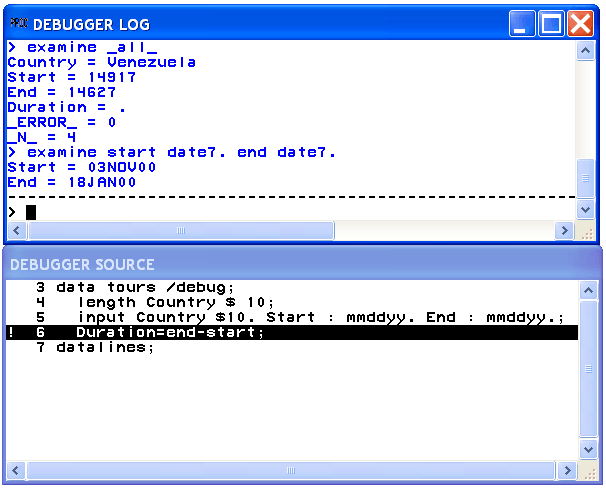
The following display shows the results:



To view formatted SAS dates, issue the EXAMINE command using the DATE**w**. format:

examine start date7. end date7.

The following display shows the results:



Because the tour ends on November 18, 2000, and not on January 18, 2000, there is an error in the variable End. Examine the source data in the program and notice that the value for End has a typographical error. By using the SET command, you can temporarily set the value of End to November 18 to see whether you get the anticipated result. Issue the SET command using the DDMMMYY**w**. format:

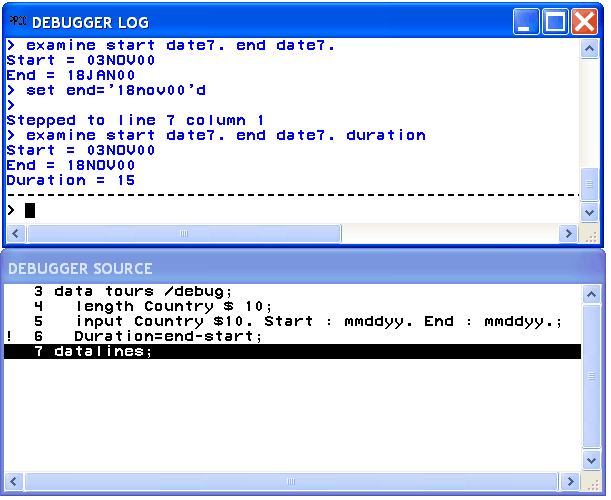
set end='18nov00'd

Press ENTER to issue the STEP command and execute the assignment statement.

Issue the EXAMINE command to view the tour date and Duration fields:

examine start date7. end date7. duration

The following display shows the results:



The Start, End, and Duration fields contain correct data.

End the debugging session by issuing the QUIT command on the DEBUGGER LOG command line. Correct the original data in the SAS program, delete the DEBUG option, and resubmit the program.

/\* corrected version \*/

options yearcutoff=1920;

data tours;

length Country $ 10;

input Country $10. Start : mmddyy. End : mmddyy.;

duration=end-start;

datalines;

Italy 033000 041300

Brazil 021900 022800

Japan 052200 061500

Venezuela 110300 111800

Australia 122100 011501

;

proc print data=tours;

format start end date9.;

title 'Tour Duration';

run;

Tour Duration 1

Obs Country Start End duration

1 Italy 30MAR2000 13APR2000 14

2 Brazil 19FEB2000 28FEB2000 9

3 Japan 22MAY2000 15JUN2000 24

4 Venezuela 03NOV2000 18NOV2000 15

5 Australia 21DEC2000 15JAN2001 25

|  |
| --- |
|  |
| **Example 3: Debugging DO Loops** |

An iterative DO, DO WHILE, or DO UNTIL statement can iterate many times during a single iteration of the DATA step. When you debug DO loops, you can examine several iterations of the loop by using the AFTER option in the BREAK command. The AFTER option requires a number that indicates how many times the loop will iterate before it reaches the breakpoint. The BREAK command then suspends program execution. For example, consider this data set:

data new / debug;

set old;

do i=1 to 20;

newtest=oldtest+i;

output;

end;

run;

To set a breakpoint at the assignment statement (line 4 in this example) after every five iterations of the DO loop, issue this command:

break 4 after 5

When you issue the GO commands, the debugger suspends execution when I has the values of 5, 10, 15, and 20 in the DO loop iteration.

In an iterative DO loop, select a value for the AFTER option that can be divided evenly into the number of iterations of the loop. For example, in this DATA step, 5 can be evenly divided into 20. When the DO loop iterates the second time, I again has the values of 5, 10, 15, and 20.

If you do not select a value that can be evenly divided (such as 3 in this example), the AFTER option causes the debugger to suspend execution when I has the values of 3, 6, 9, 12, 15, and 18. When the DO loop iterates the second time, I has the values of 1, 4, 7, 10, 13, and 16.

|  |
| --- |
|  |
| **Example 4: Examining Formatted Values of Variables** |

You can use a SAS format or a user-created format when you display a value with the EXAMINE command. For example, assume that the variable BEGIN contains a SAS date value. To display the day of the week and date, use the SAS WEEKDATE**w**. format with EXAMINE:

examine begin weekdate17.

When the value of BEGIN is 033001, the debugger displays the following:

Sun, Mar 30, 2001

As another example, you can create a format named SIZE:

proc format;

value size 1-5='small'

6-10='medium'

11-high='large';

run;

To debug a DATA step that applies the format SIZE. to the variable STOCKNUM, use the format with EXAMINE:

examine stocknum size.

For example, when the value of STOCKNUM is 7, the debugger displays the following:

STOCKNUM = medium

|  |
| --- |
| **List of Debugger Commands** |

|  |  |
| --- | --- |
| BREAK | JUMP |
| CALCULATE | LIST |
| DELETE | QUIT |
| DESCRIBE | SET |
| ENTER | STEP |
| EXAMINE | SWAP |
| GO | TRACE |
| HELP | WATCH |

|  |
| --- |
|  |
| **Debugger Commands by Category** |

|  |  |  |
| --- | --- | --- |
| ***Categories and Descriptions of Debugger Commands*** | | |
| Category | DATA Step Debugger | Description |
| Controlling Program Execution | [GO](http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/a000379432.htm) | Starts or resumes execution of the DATA step |
|  | [JUMP](http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/a000379441.htm) | Restarts execution of a suspended program |
|  | [STEP](http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/a000379462.htm) | Executes statements one at a time in the active program |
| Controlling the Windows | [HELP](http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/a000379440.htm) | Displays information about debugger commands |
|  | [SWAP](http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/a000379467.htm) | Switches control between the SOURCE window and the LOG window |
| Manipulating DATA Step Variables | [CALCULATE](http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/a000379399.htm) | Evaluates a debugger expression and displays the result |
|  | [DESCRIBE](http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/a000379413.htm) | Displays the attributes of one or more variables |
|  | [EXAMINE](http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/a000379423.htm) | Displays the value of one or more variables |
|  | [SET](http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/a000379453.htm) | Assigns a new value to a specified variable |
| Manipulating Debugging Requests | [BREAK](http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/a000379388.htm) | Suspends program execution at an executable statement |
|  | [DELETE](http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/a000379407.htm) | Deletes breakpoints or the watch status of variables in the DATA step |
|  | [LIST](http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/a000379446.htm) | Displays all occurrences of the item that is listed in the argument |
|  | [TRACE](http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/a000379468.htm) | Controls whether the debugger displays a continuous record of the DATA step execution |
|  | [WATCH](http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/a000379474.htm) | Suspends execution when the value of a specified variable changes |
| Tailoring the Debugger | [ENTER](http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/a000379418.htm) | Assigns one or more debugger commands to the ENTER key |
| Terminating the Debugger | [QUIT](http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/a000379451.htm) | Terminates a debugger session |