How to Use the Data Step Debugger

S. David Riba, JADE Tech, Inc., Clearwater, FL

ABSTRACT

All SAS® programmers, no matter if they are beginners or experienced gurus, share the need to debug their programs. Many techniques have been suggested over the years to solve the problem of debugging Data step logic. Until now SAS programmers have never been able to interactively watch the Data step execute and debug their logic as it executes.

Introduced in Release 6.11 of SAS, the Data Step Debugger is part of Base SAS software. It is an interactive tool that allows the SAS user the ability to watch and control DATA step execution.

The Data Step Debugger can be used to:

- watch program logic execute
- examine variable values as they change
- repeat program statements
- jump to other sections of the DATA step
- modify DO loops interactively
- test for "dead" code
- give commands
- observe the results interactively

The Data Step Debugger is a very useful tool, whether you are a Beginning SAS Programmer or a SAS veteran. This Tutorial will demonstrate how to use the Debugger. It will review the Debugger commands, and demonstrate how they can be used to improve the debugging process. While this is a Beginning Tutorial, the subject matter is appropriate for all levels of SAS expertise if you have not previously been introduced to the Data Step Debugger.

INTRODUCTION

The Data Step Debugger was first introduced in Release 6.11 of the SAS System. It is shipped with Releases 6.12 and 6.09e of the SAS System and is part of Base SAS Software. As its name implies, the Data Step Debugger is valid only for DATA steps and not for PROC steps.

The Data Step Debugger is easy to use and simplifies the task of debugging SAS Data steps. While It is intended to be used as an interactive debugging tool, it can be used in either Interactive or Batch mode. There are some very useful reasons for adding Debugger commands to batch jobs. Information about using the Debugger in batch mode is listed later in this paper.

INVOKING THE DEBUGGER

Invoking the Data Step debugger is as simple as adding a parameter to the DATA statement:

```
DATA dsname / DEBUG ;
```

When SAS encounters the DEBUG parameter on the DATA statement, it changes the default behavior for processing the Data step.

Normally, SAS processes a Data step in two passes. The first pass compiles the Data step. This pass performs a syntax check, resolves any macro values, and converts the program text into a program which can be executed by the computer. Assuming the Data step has compiled correctly, SAS then executes the Data step in a second pass.

Normally, the user has no control over the execution of the code that has been compiled. The default behavior is to execute the entire Data step program with no external access. Historically, the only means available to view or modify the execution of a Data step has been with hard coded PUT statements or conditional logic. However, with the DEBUG parameter, SAS first compiles the Data step and then enters DEBUG mode so the user can interact with the execution of the Data step. In DEBUG mode, the user can view and control how SAS executes the Data step program statements.

DEBUGGER WINDOWS

Once the Data Step Debugger has been invoked, an interactive environment is displayed. The Debugger environment consists of two windows -- a SOURCE window and a LOG window.

The SOURCE window displays the original Data step source code. The currently executing line is highlighted in the SOURCE window.

The LOG window is similar to the Log window in the Program Editor, with one major exception. The Debugger LOG window contains an additional command line. This command line is for issuing commands to the Debugger.

A typical Data Step Debugger session might look like this:

The top window is the Debugger Log window. The results from executing the Data step are displayed here. The Debugger command line can be found at the bottom of the Debugger Log window. The Debugger command line follows the right arrow sign at the bottom of the Log window **>** below the row of dashes.

The bottom window is the SOURCE window. Here, the Data step code is displayed, and the current line being executed is highlighted. The line numbers displayed in the Debugger Log window are the same as the line numbers displayed in the Debugger Source window.

The position and characteristics of each window can be customized, the same as the SAS Display Manager windows. Once a window has been redefined, issue the WSAVE command to permanently store those definitions.

DEBUGGER EXPRESSIONS

Since the Data Step Debugger is different from the SAS Program Editor, it has different programming requirements than the Program Editor.

The commands that are issued from the Debugger command line must follow the following rules:

- There is a limited set of Debugger commands.
- Commands that are issued during a Debugger session can include any valid SAS operator.
- SAS functions can not be used in a Debugger command.
- All commands must fit on one line. With one exception (DO blocks), Debugger expressions can not be continued on another line.
- Debugger commands can be assigned to Function Keys. This saves considerable time, as well as lessening the opportunity for errors, if frequently used commands are assigned to Debugger function keys.
- Macros can be issued from the Debugger command line.

The RECALL command will recall Debugger commands from the command stack. By default, the F4 key is assigned the value RECALL. Up to 20 previous commands can be RECALLed from the Debugger command stack.

DEBUGGER COMMANDS

There are several categories of commands that can be issued to the Data Step Debugger. These commands are issued on the Debugger Command Line, which is at the bottom of the Debugger LOG window. These commands are also available from the Debugger PMenu.

The Debugger commands can be grouped as follows:

- **Control Program Execution**
- **Manipulate Data Step Variables**
- **Manipulate Debugger Requests**
- **Tailor the Debugger**
- **Terminate the Debugger**
- **Control the Debugger Windows**

Each of these groups will be discussed in turn on the following pages.

DEBUGGER COMMANDS TO CONTROL PROGRAM EXECUTION

There are three commands that control the order of execution of a Data Step. These commands are:

- **GO**
- **JUMP**
- **STEP**

GO
Starts or resumes execution of the Data step. When a GO command is issued, execution continues until a BREAK point or WATCH variable changes, or until the Data step terminates. In addition, GO can be modified by a line number. Thus, GO 35 will continue to execute all program code until line 35, and then will stop and await the next debugger command.

GO can be shortened to G.

JUMP
Changes the point to resume execution of the Data step. JUMP requires a valid line number or label for the next program statement to execute. With the JUMP command, the normal order that a Data step is processed can be modified under user control. Sections of Data step code can be repeated, code can be skipped, conditional logic blocks can be entered or terminated, etc.

For example, to move directly to the labeled section HEADER:

```
JUMP header
```

While it is possible to do so, it is unwise to JUMP into the middle of a DO loop or into a label that is the target of a LINK - RETURN group. Directly JUMPing into either of these program groups bypasses the normal controls and unexpected results can occur.

JUMP can be shortened to J.

STEP
Executes statements one at a time. STEP can take a numeric parameter, which will execute the next X statements (i.e. STEP 4 will execute the next four statements). By default, the ENTER key steps through the program sequentially, one line at a time. However, this can be modified with the ENTER command, which will be reviewed later in this paper.

STEP can be shortened to ST

DEBUGGER COMMANDS TO MANIPULATE DATA STEP VARIABLES

There are four commands that allow the user to examine and change the values of Data step variables. These commands are:

- **CALCULATE**
- **DESCRIBE**
- **EXAMINE**
- **SET**

CALCULATE
Evaluates debugger expressions and displays the results. It is important to note that CALCULATE can not evaluate an expression that uses a function. However, calculate can be used with variable names and numeric values. The results of the calculation are displayed in the LOG window.

```
CALCULATE var1 / var2*100
```

CALCULATE can be shortened to CAL.

DESCRIBE
Displays the attributes of a variable. DESCRIBE takes as a parameter the name of a variable or _ALL_. DESCRIBE will display the Name, Type, Length, Label, Informat, and Format for a variable. If an attribute is not currently defined for the variable, it is not displayed.
The results are displayed in the LOG window.

```
DESCRIBE _ALL_
```

DESCRIBE can be shortened to DES or DESC.

EXAMINE
Displays the current value of a variable or list of variables. Like DESCRIBE, EXAMINE takes as a parameter the name of a variable or _ALL_.

```
EXAMINE var1 var2 var3 ... varn
```

In addition, the format used to display the variable value can be changed. EXAMINE can be modified with a valid format.

For example, to examine a date variable which does not have a format assigned (or to change the assigned format used for display purposes):

```
EXAMINE dob mmddyy8.
```

The results are displayed in the LOG window.

EXAMINE can be shortened to E or EX.

SET
Assigns new values to a variable. SET takes the argument Variable = Expression. The variable to be changed can also be used as part of the expression.

```
SET value = value + 1
```

Unlike the default behavior exhibited during SAS program execution, the expression to be evaluated must be the same type as the variable to be assigned. Thus, if the variable is character, the expression can not be numeric. There is no implied type conversion in the Debugger.

SET immediately changes the value of the variable, but does not display any information in the LOG window.

SET can be shortened to SE.

DEBUGGER COMMANDS TO MANIPULATE DEBUGGER REQUESTS

There are five commands that allow the user to manipulate the debugger. These commands are:

- **BREAK**
- **DELETE**
- **LIST**
- **TRACE**
- **WATCH**

BREAK
Sets Break Point to stop execution at an executable statement. Multiple Break Points can be set in a Data step. Once a Break Point is set, the debugger will continue to execute program statements until the Break condition is met. When the Break condition is met, the debugger will stop executing at the specified line and await further commands.

A more complete description of the BREAK command options follows this section.

BREAK can be shortened to B.

DELETE
Deletes a Break Point or Watch variable. Individual Break Points, Watch variables, or all Break Points or Watch variables can be deleted.

At a minimum DELETE takes one required parameter. Either BREAK (or B) or WATCH (or W) must be specified after the DELETE command. Additional parameters, as appropriate, must then be supplied. For example, _ALL_ will delete all currently defined Break Points or Watch variables:

```
DELETE W _all_
```

For Break Points, the same parameters that are used to define the Break Point can be supplied to DELETE the Break Point. These are either a line number, a statement label, or * for the current line.

For Watch variables, the name(s) of the variables will delete the Watch on each named variable.

DELETE can be shortened to D.

LIST
Displays a list of the requested items. LIST will display information about the following six types of items, if they are currently defined:

- **BREAK (or B)** current breakpoints
- **WATCH (or W)** current watch variables
- **FILES (or F)** current external files written
- **INFILES (or I)** current external files read
- **DATASETS (or D)** Input/Output datasets
- **_ALL_** values of all of the above

LIST requires one of the above arguments as a parameter.

```
LIST _all_
```

LIST can be shortened to L.

TRACE
Displays a record of program execution in the LOG window. The value of TRACE can be turned ON or OFF. By default, TRACE is set to OFF. With TRACE turned ON, the LOG window will contain a record of all actions taken during the debugging session. With no parameters, the TRACE command will display the current value (ON or OFF) of the TRACE command.

```
TRACE on
```

TRACE can be shortened to T.

WATCH
Suspends execution when the value of a variable changes. Multiple variables can be defined with one WATCH command. When the value of a Watch variable changes, the log displays the old variable value, the new variable value, and the line number. Execution of the program then stops and awaits further commands.

```
WATCH var1 var2 ... varn
```

WATCH can be shortened to W.

THE BREAK COMMAND

The BREAK command suspends execution of a Data step when a condition is met. The BREAK command can be shortened to B.

The simplest form of the BREAK command is:

```
BREAK location
```

where location can be either a line number, a statement label, or * for the current line.

Once a Breakpoint has been set, the SAS debugger will continue to execute program statements until the line (or label) is about to be executed. At that point, execution will stop and the debugger will wait for the next command.

In addition, the BREAK command can be modified by:

- **AFTER count**
- **WHEN expression**
- **DO group**

These can also be combined for a compound BREAK command.

AFTER count
Stops execution when the BREAK condition has been executed the number of times specified by COUNT. For example:

```
BREAK * AFTER 5
```

Will stop at the current line on the sixth pass.

WHEN expression
Stops execution when the WHEN expression evaluates as TRUE. If both AFTER and WHEN are specified, AFTER is evaluated first. The WHEN expression is evaluated only if the AFTER count condition is satisfied. Like the CALCULATE command, the WHEN expression can not contain SAS functions.

```
BREAK * WHEN (var = expression)
```

DO group
Like the DO statement in Base SAS, DO group processing allows multiple debugger commands to be executed. An END statement is required to terminate the sequence of commands to be executed. Unlike the other debugger commands, however, DO groups can span multiple command lines.

The simplest form of the DO group is :

```
BREAK * DO; command; command; END;
```

Multiple debugger commands can be specified between the DO and END statements. When the debugger encounters this syntax, the Data step will continue to execute until the BREAK condition is met. When the debugger suspends execution at the Break Point, the commands between the DO and END statements will be acted upon.

Multiple commands on the same line must be separated by semicolons. Commands can span multiple lines. In addition, DO groups can include IF - THEN and IF - THEN - ELSE conditional logic.

Thus, the BREAK command can be used to conditionally execute other commands as follows:

```
BREAK * DO ;
IF (condition-is-true) THEN command ;
 ELSE IF (condition-is-true) THEN DO;
  command ;
  command ;
 END ;
 ELSE command ;
END ;
```

Note the use of semicolons. Without a semicolon to terminate each command, the debugger would immediately act on the command rather than waiting on the BREAK condition. The above would be evaluated and acted upon when the BREAK condition is satisfied. At that point in the processing, the debugger would act on the conditional commands, and then wait for further commands (unless one of the commands was GO).

All of the BREAK command options can be combined as appropriate. In addition, regularly used BREAK commands can be assigned to function keys to minimize typing during a debugger session.

DEBUGGER COMMANDS TO TAILOR AND TERMINATE THE DEBUGGER

There are two commands that allow the user to tailor and terminate the Data Step Debugger. These commands are:

- **ENTER**
- **QUIT**

ENTER
Assigns one or more Debugger commands to the ENTER key. By default, the ENTER key is programmed to execute the command STEP 1. However, the ENTER key can be reprogrammed to any valid combination of debugger commands.

For example, if breakpoints and watch variables are set, it might be useful to continue execution until the program stops at a breakpoint or watch variable and then examine the values of all variables. To do this, the ENTER key can be reprogrammed to:

```
ENTER g ; e _all_
```

QUIT
Terminates the Debugger. The currently open Dataset(s) is closed, any open files are closed, and the Debugger terminates.

It is important to note that when the Data step completes execution, the Debugger environment does not automatically close. In order to exit the Debugger, the QUIT command must be issued.

QUIT can be shortened to Q.

COMMANDS TO CONTROL DEBUGGER WINDOWS

There are two global commands that can be issued. Both of these commands are not Debugger commands, but must be entered from the Command Line, Command Bar, Pmenu, or the appropriate Function Key. These commands are:

- **SWAP**
- **HELP**

SWAP
Changes the active window. The SWAP command toggles between the Debugger SOURCE and the LOG windows.

By default, Shift F3 is defined as SWAP.

HELP
Opens the Debugger command help window. All of the current SAS documentation on the Data Step Debugger is contained in the HELP file.

By default, F1 opens Debugger Help.

DEBUGGING DATA STEPS CREATED BY A MACRO

SAS default behavior in resolving Macro code creates a problem when using the Data Step Debugger. The Debugger source window displays the original source code that is being stepped through during the debug session. However, since a Macro is resolved at compile time and the source code is modified prior to execution, the original source code being executed is not available to the Debugger to display. In this case, the Debugger will still step through the Data step program code. However the SOURCE window is blank, since there is no valid source code to display.

To resolve this problem, SAS recommends a three step approach. SAS recommends that the Macro code be executed, the resolved data step code be saved, and the resolved code rerun with the Debugger turned on.

The specific steps are as follows:

- Enable the System option MPRINT
- Invoke the Macro
- Route statements in the SAS log displayed by MPRINT to a file
- Edit the file to remove line headers and extra text like notes and warnings
- Add the DEBUG option to the DATA statement
- Rerun in Debugger

There is a new system option that was introduced in Version 7 of the SAS System. The **MFILE** option routes macro **MPRINT** output to an external file. Using MFILE, the above steps would change slightly:

- Enable the System option MPRINT
- Enable the System option MFILE
- Define a fileref named MPRINT pointing to an existing external file
- Invoke the Macro
- Edit the file to remove line headers and extra text like notes and warnings
- Add the DEBUG option to the DATA statement
- Rerun in Debugger

Note that the MFILE option appends output to the external file defined in a fileref named MPRINT. The external file must exist or MPRINT will fail.

DATA STEP DEBUGGER MACROS

It is possible to define Macros which contain multiple debugger commands for use during a Debugger session. Once defined, Debugger macros are stored in the Macro library for the duration of the SAS session.

Debugger macros can also contain replaceable parameters. Thus, a single debugger macro can be defined with user definable parameters in a similar manner to the SAS macro facility.

Debugger macros can be defined with the following syntax:

```
%MACRO name; command(s); %MEND name;
```

The Macro name can be any valid SAS name. Multiple commands are separated by semicolons. All commands must be entered on the same line.

In addition to macros defined in the Debugger, the Debugger recognizes macros defined from the Program Editor. While these macros must contain valid Debugger commands, the commands do not have to be typed on a single line. The Data Step Debugger treats all macros the same, regardless if they were defined in the Program Editor or during a debugging session.

To invoke a macro in the Debugger, type the macro name from the Debugger command line.

```
%macroname
```

USING THE DISPLAY MANAGER TO ASSIGN DEBUGGER COMMANDS TO FUNCTION KEYS

The Data Step Debugger uses a separate set of Function Key definitions. In addition to the Display Manager Key definitions stored in the user's profile, the Data Step Debugger stores Function Key definitions in a keys entry named DSDKEYS. Like Display Manager key definitions, the Debugger key definitions can be customized.

To assign Debugger commands to a Function Key, prefix the command with DSD. Each command in a multiple command string must be separated by a semicolon.

For example, to define a function to Examine the values of all variables and then STep to the next program statement, a Function Key can be defined as:

```
dsd e _all_; dsd st
```

Since the Function Key definitions are stored in the user's Profile, these definitions will remain in effect until they are changed.

USING THE INTERACTIVE DEBUGGER IN BATCH MODE

Yes. The interactive Data Step Debugger can be used in batch mode. While the Debugger is intended as an interactive tool, it can be useful when running batch SAS jobs. Any Debugger command can be used, plus there is an undocumented command that is only valid in batch mode.

To run the Debugger in batch mode, the parameter on the DATA statement is slightly different. Instead of / DEBUG to invoke the Debugger, for batch mode use / LDEBUG.

```
DATA dsname / LDEBUG ;
```

This will invoke the Debugger, but will not open the interactive SOURCE and LOG windows.

When running in batch mode, the Debugger commands are entered AFTER the RUN statement at the end of the DATA step. Each command that would be typed in interactively, must be listed. It is very important to terminate the Debugger with a QUIT command. In addition, the Debugger in batch mode can not be used with a CARDS statement or a %INCLUDE statement.

Below is a sample of the Debugger commands for a batch job:

```
DATA CHECKIN / LDEBUG;
  SET HOSPITAL;
  STAY=ENDDT-STRTDT;
RUN;
GO;
WATCH ENDDT;
EX ENDDT MMDDYY10. STRTDT MMDDYY10.;
GO;
QUIT;
```

Each Debugger command is listed after the RUN statement which marks the DATA step boundary. The Debugger is terminated with the QUIT command. Between them are the commands that would have been entered interactively from the Debugger command line.

There is one additional undocumented command which is only available to the Debugger in batch mode.

❖ **PRINT**
The PRINT command takes as an argument the line number or range of line numbers from the SOURCE window. It then prints those line numbers to the Log.

WHY USE THE INTERACTIVE DEBUGGER IN BATCH MODE?

An unintended side effect of the ability to use the Data Step Debugger in batch mode is its high degree of usefulness for Source Code Control and Source Code Maintenance issues.

The traditional method of debugging SAS programs has relied on PUT statements and other external means of tracking the progress of Data step execution. Typically, when a SAS job is debugged and placed into production these extra statements are removed. The job then needs to be tested to verify that nothing was "broken" in the process of removing these extra statements. Future modifications to production jobs require the same process, and have the potential for unintended consequences when something changes that was not planned.

Using the Data Step Debugger, no code is changed. In fact, all interactions occur outside of the Data step, so maintenance and source code control are unaffected. For testing purposes, using the batch mode of the Debugger might be an ideal solution. All Debugger commands can be entered after the Data step, and each debugging session will be consistent with prior sessions.

INVOKING DEBUGGER CODE IN BATCH MODE

One additional recommendation and the Data Step Debugger becomes the ideal debugging situation when source code control is an issue. Replacing the / DEBUG parameter with a macro value allows the source code to be unchanged between test and production versions, yet allows total debugging flexibility.

For example:

```
%LET dmode = / LDEBUG ;
DATA dsname &dmode ;
```

To turn off DEBUG mode, change the macro value to:

```
%LET dmode =  ;
```

Finally, establish a macro flag to determine if the program should be debugged:

```
%LET debug = Y ;
```

Using the previous example, the completed code would now look like this:

```
%LET debug = Y ;
%LET dmode = ;
%MACRO do_debug ;
  %IF &debug = Y %THEN
    %LET dmode = / LDEBUG ;
%MEND do_debug ;
%do_debug ;

DATA CHECKIN &dmode;
  SET HOSPITAL;
  STAY=ENDDT-STRTDT;
RUN;

 WATCH ENDDT;
 GO ;
 EX ENDDT MMDDYY10. STRTDT MMDDYY10.;
 GO;
 QUIT;
```

Thus, no source code needs to be changed in order to take advantage of the Data Step Debugger whether it is interactive or batch mode. The only final step necessary before production is to remove the Debugger statements which are OUTSIDE of the production SAS code.

CONCLUSION

The Data Step Debugger is a useful tool for all SAS users, from the beginner to the advanced guru. It is an easy to user, interactive environment that allows total control of the execution stage of Data step processing. As this paper demonstrated, the Debugger can also be very useful in non-interactive batch mode. The Data Step Debugger has been part of Base SAS software since Release 6.11. This paper has reviewed the use of the Debugger and the Debugger command set.

REFERENCES

SAS Institute, Inc., *SAS® Online Doc, Version 7-1*
Cary, NC. SAS Institute Inc., 1999.

AUTHOR

The author may be contacted at:

S. David Riba
JADE Tech, Inc.
P O Box 4517
Clearwater, FL 33758
(727) 726-6099
INTERNET: dave@JADETEK.COM

This paper is available for download from the author's Web site:

HTTP://WWW.JADETEK.COM/JADETECH.HTM

ACKNOWLEDGMENTS

The author would like to gratefully acknowledge the assistance of Sharon Hamrick of SAS Institute Technical Support in reviewing this paper.

TRADEMARK INFORMATION

AUTHOR BIO

S. David Riba is CEO of JADE Tech, Inc., a SAS Institute Quality Partner who specializes entirely in applications development, consulting and training in the SAS System. Dave has presented papers and assisted in various capacities at SUGI, SESUG, NESUG, MWSUG, SCSUG, and PharmaSUG.

Dave is an unrepentent SAS bigot. His major areas of interest are efficient programming techniques and applications development using the SAS System. His SAS software product specialties are SAS/AF and FRAME technology, SAS/EIS, SAS/IntrNet, AppDev Studio, and CFO/Vision. Dave is a SAS Certified Professional.

Dave is the founder and President of the Florida Gulf Coast SAS Users Group. He chartered and served as Co-Chair of the first SouthEast SAS Users Group conference, SESUG '93, and serves as President of the Executive Council of the SouthEast SAS Users Group. He also serves on the Executive Council of CONSUG, the Consultant's SAS Users Group. His first SUGI was in 1983, and he has been actively involved in both SUGI and the Regional SAS User Group community since then.

Dave is currently the Co-Chair of SSU 2001, the combined SouthEast and SouthCentral SAS Users Group conference to be held in New Orleans in August, 2001.