

# Jump-Starting the Interactive Debugger

Mike Atkinson, Acko Systems Consulting Inc

## ABSTRACT

It is usually fairly straightforward to investigate problems with the SAS data step. The information in the log, and the ability to look at datasets between each step are, in my opinion, great strengths of SAS. For those relatively rare problems that can't be solved by reviewing the log or intermediate results, there is the interactive debugger. A straightforward subset of the commands in the interactive debugger can help pinpoint thorny problems.

Generally when I start to use the interactive debugger, I've already identified at least one data value for which the processing appears to have gone awry. Before starting the debugger, I add a few statements to the code to help with jump-starting the debugger. Then, once I've started the interactive debugger, it is easy to jump forward through to data near the point at which the error occurred.

The interactive debugger can step through statements one at a time so I can see which path in the code is being followed. And I can view the contents of the variables at any point without needing to add debugging PUT statements.

## INTRODUCTION

A relatively small subset of the commands available in the interactive debugger can be put to great use:

```
break <statement-number>
delete break <statement-number>
enter step
ex _all_
ex <variable-name>
go
step
quit
```

There are other features available in the interactive debugger, but I find that the above meet my needs most of the time.

## A CASE STUDY

Imagine processing a set of data representing medical claims. I want to determine when a claim has occurred for a patient within 12 months or within 24 months of the previous claim for the same patient. Suppose this was the first cut at a solution:

```
proc sort data=claims;
    by ptnt_num service_date;
run;

*;
* Find if two records are within one year or within two years ;
* of the previous medical record for the same patient;
*;

data flag_12_or_24_mons;
    keep ptnt_num service_date fees_item within_12_mon within_24_mon;
    format service_date yymmdd10.;
    retain within_12_mon within_24_mon prev_service_date;

    set claims;
        by ptnt_num service_date;

        * Initialise on a new patient;
        if (first.ptnt_num) then do;
            prev_service_date = mdy(12, 31, 9999);
            within_12_mon = 0;
            within_24_mon = 0;
        end;

        * On other records, make date range check;
        else if (service_date ne prev_service_date) then do;
            * Calculate number of months between the two dates;
            months_later = intck('month', prev_service_date,
                                service_date);
            if (day(prev_service_date) < day(service_date)) then
                months_later = months_later + 1;

            * Set flags indicating when date falls within each range;
            if (months_later <= 12) then
                within_12_mon = 1;
            if (months_later <= 24) then
                within_24_mon = 1;
        end;

        prev_service_date = service_date;
run;
```

This code is meant to mark claims that are within 12 and within 24 months of each other. The variable `within_12_mon` is to be set to 1 (TRUE) if a claim is within 12 months of the previous claims for the same patient and 0 (FALSE) otherwise. Similarly for the variable `within_24_mon` and a 24 month period.

The code seems correct, but sometimes it gets the wrong answer. I've looked at the log, at the data prior to and after this step, and still can't figure out where it went wrong.

This code would usually be run with a large number of claims, but suppose the following small set of sample data formed the input:

claims dataset

ptnt_num	service_date	fee_item
122345	April 11, 2003	00100
201201	June 25, 2002	00100
201201	December 2, 2002	00111
222256	September 6, 2003	00100
222256	September 8, 2004	00120
222256	January 4, 2005	00100
312345	April 12, 2002	00100
312345	July 7, 2002	00100
312345	March 15, 2004	00200

When this code is run, it generates the following results. This is mostly correct, but there is one wrong value for the variable named `within_12_mon`, indicated in red.

ptnt_num	service_date	within_12_mon	within_24_mon	fee_item
122345	April 11, 2003	0	0	00100
201201	June 25, 2002	0	0	00100
201201	December 2, 2002	1	1	00111
222256	September 6, 2003	0	0	00100
222256	September 8, 2004	0	1	00120
222256	January 4, 2005	1	1	00100
312345	April 12, 2002	0	0	00100
312345	July 7, 2002	1	1	00100
312345	March 15, 2004	1	1	00200

RATS, THE PROBLEM ISN'T OBVIOUS

The problem is somewhere in the data step, but where? I've checked the log, I've looked at the data before and after the data step, and nothing is jumping out. I might try a couple of things like commenting out the keep statement, so that more variables come through into the output dataset. And putting a date format on `prev_service_date` seemed like it might help, but after I do that it turns out I'm still stumped.

This seems like a good time to try the interactive debugger. I will go through the code step by step, seeing which statements get executed, and examining the values of variable at any point.

Using the interactive debugger can be relatively time consuming, however, so it's not usually my first approach to debugging. When I do use the interactive debugger, I have a technique to jump-start to the data I want to look at.

## JUMP-STARTING THE DEBUGGER

Before I run the debugger, I almost always add a couple of statements to the data step just to aid with debugging. I add an IF statement, usually just after the SET statement, testing for a specific value that I know is related to the problem. Following the IF statement, I add another statement – this second statement is the one that will be used as the break point. The IF statement itself is not particularly useful as a break point, since the IF statement is executed whether its condition evaluates as true or false.

With the extra statements in place, I'll be able to quickly set a breakpoint within the interactive debugger and then jump through the data to an observation of interest.

To start the debugger, it is also necessary to add the debug keyword to the data statement.

```
data flag_12_or_24_mons / debug;
  *keep ptnt_num service_date fee_item within_12_mon within_24_mon;
  format service_date prev_service_date yymmdd10.;
  retain within_12_mon within_24_mon prev_service_date;

  set claims;
    by ptnt_num service_date;

  if (ptnt_num = 312345) then do;
    put 'debugging' ptnt_num =;
  end;

  * Initialise on a new patient;
  if (first.ptnt_num) then do;
    prev_service_date = mdy(12, 31, 9999);
    within_12_mon = 0;
    within_24_mon = 0;
  end;

  * On other records, make date range check;
  else if (service_date ne prev_service_date) then do;
    * Calculate number of months between the two dates;
    months_later = intck('month', prev_service_date,
      service_date);
    if (day(prev_service_date) < day(service_date)) then
      months_later = months_later + 1;

    * Set flags indicating when date falls within each range;
    if (months_later <= 12) then
      within_12_mon = 1;
    if (months_later <= 24) then
      within_24_mon = 1;
  end;

  prev_service_date = service_date;
run;
```

## IN THE DEBUGGER

The debugger opens two windows, but may initially fill the screen with one of them. To see both windows, I click the grey resize button at top right (☐). I can now see both the Debugger Log window and the Debugger Source window, with execution paused at the first executable statement.

My next step is to place a break on the statement following the IF statement – in this case, the PUT statement just added. This statement must be referred to by number, shown to the left of the statement in the Debugger Source window. For example, to add a break at statement 100:

```
break 100
```

The break placed on the statement numbered 100 should now be indicated in the Debugger Source window with an exclamation mark to the left of the line of code.

Once I've done this, the `go` command will start execution until either a statement with a break is reached or until the data step finished processing the last observation. Hopefully it will stop at my added PUT statement (with the break), or I'll need to wait until all the observations are processed and try again.

Once the statement with the break is reached, I'll get a chance to issue another command in the Debugger Log window. For instance, I can check that I've found the right data by typing the `examine` command (or `ex` for short). I can see the value of a single variable, e.g.

```
ex ptnt_num
```

or the values of all variables:

```
ex _all_
```

I can now execute statements one by one, using the `step` command. Or I could set a different breakpoint. For instance, I could set a breakpoint near to the statements where variable `within_12_mon` is calculated and jump to that point with the `go` command again. In this case, I will probably delete the break that is already in place.

For example:

```
break 113
```

```
delete break 100
```

```
go
```

I use the `step` command quite a bit. Once I believe I am in the vicinity of where the problem occurred, I step through the statements one by one looking for unexpected branches in the logic. To reduce typing, it may be useful to assign the `step` command to the enter key:

```
enter step
```

After this, I can tap the enter key (when no debugging command has been typed in) to step forward to the next executable statement.

To solve my “within 12 months” problem, I go (or step) through the code until I reach the observation of interest – the date of March 15, 2004 for patient ID 312345. At this point I go slowly, checking values of variables now and then.

The data step gets to the IF statement testing for within 12 months, and, correctly, the assignment statement is not executed, since this date is 18 months after the previous date. I check the value of the variable using the command:

```
ex within_12_mon
```

What? It is set to 1 (TRUE), even though the IF statement did not execute! That's not right – what is going on here?

After scratching my head few times I realise that the value of 1 has been RETAINED from a previous observation for the same patient. Why do I need to retain this variable anyway? The variable named prev\_service\_date needs to be retained, since we need to see the values from previous observations. But I don't need a retain for within\_12\_mon or within\_24\_mon – I'll remove those from the retain statement and try again. I also realise that I'll need an ELSE statement following the IF statement, to set within\_12\_mon to 0 if it is not being set to 1.

To get out of the debugger, I enter the command:

```
quit
```

DONE

I make the change to the RETAIN statement, add a couple of ELSE statements, and run the data step (outside the debugger) again. Yes! It works.

This was a problem that might have been difficult to find without the interactive debugger.

### Contact Information

Mike Atkinson  
Acko Systems Consulting Inc  
[acko@telus.net](mailto:acko@telus.net)  
381-1077  
208-6908 (cel)