SAS seminar

2003-06-03

Åsa Klint

The little SAS book

Chapter 8

" Debugging your SAS programs"

By LD Delwiche and SJ Slaughter

Writing SAS Programs That Work

Make programs easy to read

- put only one SAS statement on each line
- Use indention to show the different parts of a program

```
Data mydata;
  set libname.mydata;
        if sex=1 then do;
                start=age_menarche;
                :
        end;

        else if sex=0 then do;
                :
                :
        end;
Run;
```

- Use comment statements generously to document your programs

/* comment */ alternatively * comment ;

Test each part of the program

Increase your programming efficiency by making sure each part of the program works before moving on to the next part.

Test programs with small data sets

Use options OBS= and FIRSTOBS= to select a subset of the full data set. (time-saving if you have a large amount of data)

Be observant of the colors in your programming. The Enhanced editor color-codes your programming as you write it which makes it easy to discover missing semicolons etc as the rest of your programming will appear in 'wrong' colors.

For example, SAS keywords appear in blue, comments in green and all text within quotes appear in purple.

• Fixing Programs That Don't Work

Read the SAS log which gives three types of messages:

1. errors (in red)

   Usually comes from a syntax or spelling mistake.

   The location of the error is usually easy to find as it is underlined but it is not necessary the source of the error (this could be earlier in the program)

2. warnings (in green)

   The program still works with warnings but SAS might have done something you didn't want. Read them carefully and make sure you know what they are about and that you agree with them.

3. notes (in blue)

   Sometimes just a piece of information, other times an indicator of a problem. Read carefully!

Look for common mistakes first

For example, you may receive an error message saying the SET statement is not valid which may seem unlogic as you know this is a valid statement. In cases like this: look for missing semicolons in the previous statement!

SAS reads statements from one semicolon to the next without regard to the layout of the program so if you leave out a semicolon, essentially you concatenate two SAS statements.

- The DATASTMTCHK system option

A control command for what names you can use for SAS data sets in a DATA statement to prevent you from accidentally overwrite a data set. By default the words MERGE, RETAIN, SET and UPDATE cannot be used as data set name but all SAS keywords can be made invalid SAS data set names by writing

OPTIONS DATASTMTCHK=ALLKEYWORDS;

Example

missing semicolon

```
data diet
 set seminar.diet_raw2;
    if job='conductor' then do;
     /*creating new variables */
           x=10 ;
           z='tio';
    end;
run;
```

program-file

```
2    data diet
NOTE: SCL source line.
3      set seminar.diet_raw2;
       ---
       56
ERROR 56-185: SET is not allowed in the DATA statement when option
    DATASTMTCHK=COREKEYWORDS.
          Check for a missing semicolon in the DATA statement, or use
    DATASTMTCHK=NONE.
```

log-file

Common notes and errors

- Note: Invalid data
- Note: Missing values were generated
- Note Numeric values have been converted to character (or vice versa)
- Error: invalid option
- Error: the option was not recognized
- Error: statement is not valid
- Note: variable is uninitialized
- Error: variable not found

- Note: Invalid data

  The invalid-data note appears when SAS is unable to read from a raw data file because the data are inconsistent with the INPUT statement.

  A common mistake is to type 0 instead of the letterO (or vice versa). If the variable is numeric then SAS is unable to interpret the letter O.
- In this case SAS does two things; it sets the value of this variable to missing and prints out this message for the problematic observation.
- Reading the SAS log reveals where the problem occurred (line number and column) and two new variables: _ERROR_ (=1 if there is a data error for that observation) and _N_ (number of times SAS has looped through the data step)

  Example

```
NOTE: Invalid data for id in line 1 1-3.
1        0o7 BOND SA 11
id=. name=BOND class=SA _ERROR_=1 _N_=1
```

Possible causes for this note:

- Using the letter O instead of 0, letter l instead of 1 etc.
- Forgetting to specify that the variable is character (SAS assumes numeric)
- Incorrect column specifications producing embedded spaces in numeric data.
- Using wrong informat such as MMDDYY. instead of DDMMYY.
- Invalid dates (such as September 31) read with date informat.

- Note: Missing values were generated

This note appears when SAS is unable to compute the value of a variable as a result of missing values in your data.

Example (variables: id, length, wgt1, wgt2, wgt3):

```
100 176 65.1 69.0 69.9
101 165 75.3 77.3
102 168 64.6 59.1 58.3
103 159 64.0 63.0 64.3
```

Say you want to calculate the mean weight for each person and you give SAS the following command

```
weight_avg=(wgt1+wgt2+wgt3)/3;
```

However, this way missing values would be generated for all observations with at least one missing value on the weight-variables.

Using the SAS function MEAN instead would only create missing if there were observations without any information on weight.

```
weight_avg=MEAN(wgt1,wgt2,wgt3);
```

- Note: Numeric values have been converted to character (or vice versa)

There are two types of variables: numeric and character.

If you mix up the type of the variable, SAS tries to fix your program by converting variables as needed.

However, a good rule to follow is that if variables need to be converted it is preferable to, explicitly, do it yourself so you know and are in control of what has been done.

Converting commands:

Char → num *new_numvar=input(old_charvar, informat)*

Num → char *new_charvar=put(old_numvar, format)*

Example of format/informat: best8.

Example

Program-file

```
data myfile;
    varA=5;   /* numeric variable */
    varB='5'; /* character variable */
    varC=varA+varB;
    varD=6501235596; /* personnummer stored as numeric variable */
    varE=substr(varD,1,6); /* using the SAS function SUBSTR for
          character variables the extract the birthdate */
run;
```

Log-file

```
NOTE: Character values have been converted to numeric values at the places given by:
 (Line):(Column).
     11:15
NOTE: Numeric values have been converted to character values at the places given by:
(Line):(Column).
     13:17
```

When SAS cannot make sense out of your statements it stops executing the current DATA or PROC step and prints one of these messages

- Error: invalid option
- Error: the option is not recognized
- Error: Statement is not valid

The first two mean that the statement is valid but SAS can't make sense out of an apparent option.

The last means that SAS can't make sense out of it at all.

However, SAS underlines the part where it got confused!

Possible causes:

- a misspelled keyword
- A missing semicolon
- A DATA step statement in a PROC step (or vice versa)
- The correct option with the wrong statement
- An unmatched quote
- An unmatched comment

- NOTE: Variable is uninitialized.

  or
- Error: Variable not found

In a DATA step: SAS creates the variable and continues to execute the program. However, if nothing else is stated the variable gets a missing value for all observations which might not be what you wanted.

In a PROC step: depending on how critical the statement is, SAS prints the error message and either stops the procedure or attempts to run the step.

Possible causes:

- Misspelling a variable name
- Using a variable that was dropped at some earlier time
- Using the wrong data set
- Committing a logic error, such as using a variable before it is created

Try using PROC CONTENTS to see a list of variables

- DATA step produces wrong results but no error message

The more complex your programs are, the more likely you are to get this type of error. That is it is not really an error, somewhere SAS got the wrong instructions.

Example

Using the data set I introduced previously but given that there were no missing values. (id, length, wgt1, wgt2, wgt3)

```
100 176 65.1 69.0 69.9
101 165 75.3 77.3 78.4
102 168 64.6 59.1 58.3
103 159 64.0 63.0 64.3
```

```
data lowweight;
  set weight;
    avg_weight=MEAN(wgt1+wgt2+wgt3);
    if avg_weight<65;
run;
```

Running this program gives a data set with 0 observations which is not what we expected after looking at the data!

Using the PUT statement to debug

When using the PUT statement, SAS writes the data in the log window which can be useful for debugging.

example:

PUT _ALL_; SAS will print all the variables and all data values in your data set.

If you have many variables you can print a selection of variables by specifying them.

PUT varname1= varname2= ;

```
data lowweight;
  set weight;
    avg_weight=MEAN(wgt1+wgt2+wgt3);
    PUT wgt1= wgt2= wgt3= avg_weight= ;
    if avg_weight<65;
run;
```

...which results in the following log file where you can see that there is something wrong with the variable avg_weight

```
wgt1=65.1 wgt2=69 wgt3=69.9 avg_weight=204
wgt1=75.3 wgt2=77.3 wgt3=78 avg_weight=230.6
wgt1=64.6 wgt2=59.1 wgt3=58.3 avg_weight=182
wgt1=64 wgt2=63 wgt3=64.3 avg_weight=191.3
NOTE: There were 4 observations read from the data set WORK.WEIGHT.
NOTE: The data set WORK.LOWWEIGHT has 0 observations and 6 variables.
NOTE: DATA statement used:
      real time          0.06 seconds
      cpu time           0.06 seconds
```

In this case the arguments were incorrectly specified in the MEAN function. SAS was instructed to take the mean of the sum of the three weights (which is the mean of one number) and did so but this was not what we wanted...

- SAS stops in the middle of a job

  This is not necessarily a SAS problem but rather the operating environment stopping your program. It could also be a programming error preventing SAS from seeing the entire job.

  possible causes:

- An unmatched quote or comment
  SAS stops as it simply believes the remainder of the program is part of a quote or comment
  solution: submit commands that will close the quote/comment
  (i. e. **' ; run;** or ***/ ; run;**)
- No RUN statement at the end of a program

- SAS runs out of memory or disk space

This kind of problem occur for example when you sort a large data set with many variables, do intensive computations or create many large temporary data sets during the course of a SAS session

Reduce disk space by decreasing the number of bytes needed for each variable. However this does not help memory problems as all numbers are expanded to the fullest precision when processing the data.

You can change the length of variables in existing SAS data sets by using a LENGTH statement between a DATA statement and a SET, MERGE or UPDATE statement (ex LENGTH varname $ 4 ;)

Some advice:

- If you only intend to use a subset of your data set, use the subsetting IF statement as soon as possible.
- Use DROP= or KEEP= to reduce the number of variables in your data set
- Clear the output- and the log-window often.
- Use the COMPRESS option. Check the log window for information on the change in size of your data set (sometimes COMPRESS increases the size of your data set!)
  example

```
data weight_compr (compress=yes);
      set weight;
run;
```

Final message:

Don't ignore warnings, notes or error messages because you don't understand what they mean.

"problems that go away by themselves come back by themselves"

Thank you for listening!

Next seminar: June 17th