Paper CT01

I love SAS logs (and You Can Too)

Sarah Berenbrinck, Independent, UK

ABSTRACT

ERRORs are red
NOTEs are blue
WARNINGs are green
Assuming we're using the same SAS colour scheme

I am the kind of programmer who uses trial and improvement as my method of choice. My A-level maths teacher taught it as a reasonable method for solving problems, so now I'm owning it.
That means that I need to clearly see where improvements are needed to get my code both right and efficient.
I find that SAS® logs allow for this. I have used a few other programming languages and I mostly find the logs to be along the lines of "computer says noooo", but SAS tells me why it can't do something, so I know exactly how to go fix it.

INTRODUCTION

I will review some common errors, warnings and notes including fixes, with some other useful things from SAS logs.

DATA (FOR LATER)

Here is some code to create a dataset (using datalines) which will be used to demonstrate some things later in this paper.

```
data phuse1 ;
  input year 1-4 city $ 6-14 country $ 16-30 ;
datalines ;
2016 Barcelona Spain
2017 Edinburgh Scotland
2018 Frankfurt Germany
2019 Amsterdam The Netherlands
;
run ;
```

THE ERRORS THAT ARE CLEAR

These are the reason I love SAS logs.

THIS VARIABLE DOESN'T EXIST

Code tries to use a variable that's not in the dataset.

```
proc print data=phuse1 ;
  var place ;
run ;
```

This causes an error as the proc can't run and it is very clear what the issue is.

```
457       +proc print data=phuse1 ;
458       +  var place ;
ERROR: Variable PLACE not found.
459       +run ;

NOTE: The SAS System stopped processing this step because of errors.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.01 seconds
      cpu time            0.00 seconds
```

This can be updated to use variables that exist in the dataset or by removing var and printing all variables.

```
proc print data=phuse1 ;
run ;
proc print data=phuse1 ;
  var year city ;
run ;
```

THE FORMAT DOESN'T EXIST

With SAS option fmterr or nofmterr you can control whether using a non-existing format results in an error or just a note. Option fmterr is default.

```
data phuse2 ;
  set phuse1 ;
  new=input(year,test.) ;
run ;
```

```
561       +data phuse2 ;
562       +  set phuse1 ;
563       +  new=input(year,test.) ;
                           _____
                           48
ERROR 48-59: The informat TEST was not found or could not be loaded.

564       +run ;

NOTE: Numeric values have been converted to character values at the places given by: (Line):(Column).
      563:13
NOTE: The SAS System stopped processing this step because of errors.
WARNING: The data set WORK.PHUSE2 may be incomplete.  When this step was stopped there were 0 observations and 3 variables.
WARNING: Data set WORK.PHUSE2 was not replaced because this step was stopped.
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

Here we get an error and a warning, it is best practice to start solving the issues in the order they appear in and typically this will also fix the later issues.

This should be corrected by using a format that exists, (this could be that the format you are using needs to be created), although you can suppress the error, so you only get a note by changing the option. This does not fix the issue, but does allow the datastep to run.

```
options nofmterr ;
data phuse2 ;
  set phuse1 ;
  new=input(year,test.) ;
run ;
```

```
1212      +data phuse2 ;
1213      +  set phuse1 ;
1214      +  new=input(year,test.) ;
                           _____
                           485
NOTE 485-185: Informat TEST was not found or could not be loaded.

1215      +run ;

NOTE: Numeric values have been converted to character values at the places given by: (Line):(Column).
      1214:13
NOTE: There were 4 observations read from the data set WORK.PHUSE1.
NOTE: The data set WORK.PHUSE2 has 4 observations and 4 variables.
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

EXPECTING A … USEFUL FOR MAKING LESS COMMON PROCS WORK

You run a proc freq and the output is what you are looking for, so now you want to output these results to a dataset

```
proc freq data=phuse1 ;
  tables year / output=phuse3 ;
run ;
```

```
1215      +proc freq data=phuse1 ;
1216      +  tables year / output=phuse3 ;
                          ______
                          22
                          76
ERROR 22-322: Syntax error, expecting one of the following: ;, AGREE, ALL, ALPHA, BDT, BIN, BINOMIAL, CELLCHI2, CHISQ, CL, CMH,
              CMH1, CMH2, COMMONRISKDIFF, COMONMRDIFF, CONTENTS, CONVERGE, CROSSLIST, CUMCOL, DEVIATION, EXACT, EXPECTED, FISHER,
              FORMAT, GAILSIMON, GS, JT, KAPPA, LINE, LIST, MAXITER, MAXLEVELS, MEASURES, MISSING, MISSPRINT, NOCOL, NOCUM, NOFREQ,
              NOPERCENT, NOPRINT, NOROW, NOSPARSE, NOWARN, ODDSRATIO, OR, OUT, OUTCUM, OUTEXPECT, OUTPCT, PEARSONRES, PEARSONRESID,
              PLCORR, PLOTS, PRINTKWTS, PRINTWTS, RELRISK, RISKDIFF, SCORE, SCORES, SCOROUT, SPARSE, STDRES, STDRESID, TABLE,
              TESTF, TESTP, TOTPCT, TREND, WARN.
ERROR 76-322: Syntax error, statement will be ignored.
1217      +run ;
```

SAS lists all the options that can be used after the / in the tables statement, and in this case, we can see that `output` needs to be corrected to `out`.

```
proc freq data=phuse1 ;
  tables year / out=phuse3 ;
run ;
```

```
NOTE: There were 4 observations read from the data set WORK.PHUSE1.
NOTE: The data set WORK.PHUSE3 has 4 observations and 3 variables.
NOTE: The PROCEDURE FREQ printed page 4.
NOTE: PROCEDURE FREQ used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

Some parameters go in the proc-statement, and when I add an incorrect option, SAS suggests all possible options, including `noprint` that I should have used here.

```
1376      +proc freq data=phuse1 print ;
                                 _____
                                 22
                                 202
ERROR 22-322: Syntax error, expecting one of the following: ;, (, COMPRESS, DATA, FC, FORMCHAR, NLEVELS, NOPRINT, ORDER, PAGE.
ERROR 202-322: The option or parameter is not recognized and will be ignored.
1377      +  tables year / out=phuse3 ;
1378      +run ;

NOTE: The SAS System stopped processing this step because of errors.
WARNING: The data set WORK.PHUSE3 may be incomplete.  When this step was stopped there were 0 observations and 0 variables.
WARNING: Data set WORK.PHUSE3 was not replaced because this step was stopped.
NOTE: PROCEDURE FREQ used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

LIBNAME NOT ASSIGNED

So far, we have only used data in the default location, which is called the work library. The work library can be used implicitly as we have done above with 1-level dataset names, or explicitly by using a 2-level dataset name where the first level is the library, `work.phuse1`. Often you will use data from different locations. To read from different locations use a different library and the reference for this library is called a libname.

```
data phuse4 ;
  set myinput.phuse1 ;
run ;
```

If the libname doesn't exist, you get an error.

```
414       +data phuse4 ;
415       +  set myinput.phuse1 ;
ERROR: Libref MYINPUT is not assigned.
416       +run ;

NOTE: The SAS System stopped processing this step because of errors.
WARNING: The data set WORK.PHUSE4 may be incomplete.  When this step was stopped there were 0 observations and 0 variables.
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

It may be that you spelt the libname wrong, still need to assign the libname, or - in our case - use the libname work. To define a libname, you need a libname statement.

```
libname myinput "users/data/phuse" access=readonly ;
```

DATA NOT SORTED CORRECTLY

If you want to use a by variable, then the data needs to be sorted.

```
data phuse5 ;
  set phuse1 ;
  by country ;
  if first.country then x=city ;
run ;
```

```
ERROR: BY variables are not properly sorted on data set WORK.PHUSE1.
year=2016 city=Barcelona country=Spain FIRST.country=1 LAST.country=1 _ERROR_=1 _N_=1
NOTE: The SAS System stopped processing this step because of errors.
NOTE: There were 2 observations read from the data set WORK.PHUSE1.
WARNING: The data set WORK.PHUSE5 may be incomplete.  When this step was stopped there were 0 observations and 3 variables.
NOTE: DATA statement used (Total process time):
```

It may be that you used the incorrect variable name and should be using year which is already sorted, but it is more likely that you need to sort your data first.

```
proc sort data=phuse1 out=phuse6 ;
  by country ;
run ;
```

I should note here that proc sql can be used to sort the data as it is read in and that is also a possible fix.

THE ERRORS THAT ARE ANNOYING

When you are familiar with SAS these errors are easy to spot and fix, but the descriptions in the log are not as clear.

MISSING SEMICOLON

You may miss a semicolon when typing.

```
proc sort data=phuse1 out=phuse6 ;
  by country
run ;
```

```
1986      +proc sort data=phuse1 out=phuse6 ;
1987      +  by country
1988      +run ;
ERROR: Variable RUN not found.
```

SAS doesn't know that you have missed a semi colon and assumes run is a variable, if you have laid out your code clearly, it is fairly easy to spot why the error was reported, although the error message is not as clear as in the examples above.

NON-MATCHED QUOTES

```
* in base comments it's not a problem to use (pairwise or not) quotes ;
%* in macro comments it's a problem to use (non-pairwise) quotes ;
%* in macro comments it's not a problem as long as quotes ' occur pairwise ;
```

The error can appear many lines below in the log and be hard to identify. I find it best to not use apostrophes in comments as SAS reads them as unclosed quotes however if they are needed then surround them with double quotes.

WARNINGS

Warnings do not force SAS to stop running but are cause for concern, so they should be investigated and prevented.

VARIABLE LENGTH DIFFERENCES

When merging data, the variables may have different lengths.

For this we create a dataset phuse1a where the variable called city is longer (code as before with a change to the length statement).

```
  length year 8 city $12 ;
```

Then merge or set the 2 datasets together.

```
data phuse10 ;
  merge phuse1 phuse1a ;
  by year ;
run ;

data phuse11 ;
```

```
  set phuse1 phuse1a ;
  by year ;
run ;
```

Both give the same warning:

```
WARNING: Multiple lengths were specified for the variable city by input data set(s). This can cause truncation of data.
```

This issue is due to how the PDV (program data vector) works, which defines the variables and their attributes, based on the earliest variable declaration encountered at datastep compilation.

There are a couple of ways to prevent this warning.

You could add a length statement at least as long as the longer variable

```
data phuse12 ;
  length city $15 ;
  merge phuse1 phuse1a ;
  by year ;
run ;
```

Or put the dataset with the longer variable first, however this only works if all the common variables are longer in the first dataset.

```
data phuse13 ;
  set phuse1a phuse1 ;
  by year ;
run ;
```

MISSPELT KEYWORDS

Sometimes you may misspell a keyword.

```
proc print date=phuse1 (where=(year=2019)) ;
run ;
```

In this case the proc still runs but you get a warning and need to fix the spelling.

```
625       +proc print date=phuse1 (where=(year=2019)) ;
                      ____
                      1
WARNING 1-322: Assuming the symbol DATA was misspelled as date.
626       +run ;
```

Other keywords are more vital and must be correct for the procedure to run.

```
proc prnt data=phuse1 (where=(year=2019)) ;
run ;
```

```
704       +proc prnt data=phuse1 (where=(year=2019)) ;
ERROR: Procedure PRNT not found.
705       +run ;
```

```
proc print data=phuse1 (were=(year=2019)) ;
run ;
```

```
783       +proc print data=phuse1 (were=(year=2019)) ;
                                  ____
                                  22
ERROR 22-7: Invalid option name WERE.
784       +run ;
```

NOTES OF CONCERN

I will describe some notes which stop SAS logs being considered “clean”.

THIS VARIABLE DOESN’T EXIST

Code tries to use a variable that’s not in the dataset.

```
data phuse2 ;
  set phuse1 ;
  new=date ;
run ;
```

gives a note:

```
NOTE: Variable date is uninitialized.
```

This means there is a problem with your code and while there is no error this needs to be fixed.

NUMERIC/CHARACTER CONVERSIONS

If you assume that a character variable is a number in your code it will work but output a note that is bad practice to have.

```
data phuse2 ;
  set phuse1 ;
  new=input(year,4.) ;
run ;
```

```
NOTE: Numeric values have been converted to character values at the places given by: (Line):(Column).
      648:13
```

Knowing what `input` and `put` expect and generate can help when fixing these notes.

`input` expects the incoming value to be a character and can output a character or numeric.

`put` accepts character or numeric as the incoming value and outputs a character.

MERGES

We must be careful with merges as this code runs fine with no odd notes but there is no control in how the datasets combine.

```
data phuse7 ;
  merge phuse1 phuse3 ;
run ;
```

```
NOTE: There were 4 observations read from the data set WORK.PHUSE1.
NOTE: There were 4 observations read from the data set WORK.PHUSE3.
NOTE: The data set WORK.PHUSE7 has 4 observations and 5 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

Below is much better code for the task.

```
data phuse8 ;
  merge phuse1 phuse3 ;
  by year ;
run ;
```

It is rare for merges with no by statement to be correct, therefore it is helpful to use the following option:

`options mergenoby=warn ;` as the default is `nowarn`.

This code gives a note, which would need to be investigated. In the example below the datasets are identical so there are no consequences, however if this was birthdates and names and the different datasets had different people with the same birthdate then data would be lost.

```
data phuse1b phuse1c;
  length year 8 city $9 ;
  input year city $ ;
datalines ;
2016 Barcelona
2017 Edinburgh
2018 Frankfurt
2019 Baltimore
2019 Amsterdam
;
run ;

data phuse14 ;
  merge phuse1b phuse1c ;
  by year ;
run ;
```

```
NOTE: MERGE statement has more than one data set with repeats of BY values.
NOTE: There were 5 observations read from the data set WORK.PHUSE1B.
NOTE: There were 5 observations read from the data set WORK.PHUSE1C.
NOTE: The data set WORK.PHUSE14 has 5 observations and 2 variables.
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

This code has what I would consider a clean log, but an incorrect dataset.

```
data phuse15 ;
  merge phuse1 phuse1b ;
  by year ;
run;
```

```
NOTE: There were 4 observations read from the data set WORK.PHUSE1.
NOTE: There were 5 observations read from the data set WORK.PHUSE1B.
NOTE: The data set WORK.PHUSE15 has 5 observations and 3 variables.
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

	year	city	country
1	2016	Barcelona	Spain
2	2017	Edinburgh	Scotland
3	2018	Frankfurt	Germany
4	2019	Baltimore	The Netherlands
5	2019	Amsterdam	The Netherlands

PLOTS WITH PARTS CUT OFF

Our dataset has 4 years however we can set the axis so only 2 are plotted.

```
axis1 order=(2017 to 2018 by 1) ;
proc gplot data=phuse1 ;
   plot year * city / vaxis=axis1 ;
run ;
quit ;
```

```
324        +axis1  order=(2017 to 2018 by 1) ;
325        +proc gplot data=phuse1 ;
326        +   plot year * city / vaxis=axis1 ;
327        +run;

NOTE: 2 observation(s) outside the axis range for the year * city request.
```

In this case we are missing data (and perhaps intentionally), however if you are plotting with explicit order statements that you set from test data, you may be losing data when a different dataset is used.

To create robust code, create macro variables of the maximum and minimum values and use these to populate the axis, or let SAS generate the axis.

HOW TO CREATE YOUR OWN ERRORS AND WARNINGS

You can add your own errors, warnings or comments to the log to aid with review.

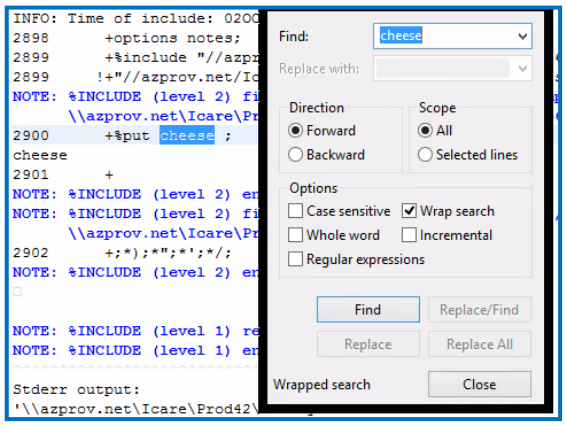
Below is a best practice way to add them to your log, this code is dynamic, so the log only shows errors if the condition is met, the error and warning strings are broken up in the code so that they don't show up in a text search is the condition isn't met. There are variables in the printed text so the log has more detail to help identify the issue.

```
data _null_ ;
  set phuse1 ;
  if year=2017 then do ; put "ERR" "OR: panic, there is an issue city=" city ; end ;
  if year=2018 then do ; put "WAR" "NING: oooo, green text city=" city ; end ;
  if year=2019 then do ; put "NOTE: something to investigate city=" city ; end ;
  if year=2020 then do ; put "ERR" "OR: nothing to see here city=" city ; end ;
run;
```

```
ERROR: panic, there is an issue city=Edinburgh
WARNING: oooo, green text city=Frankfurt
NOTE: something to investigate city=Amsterdam
```

Note the use of `put` in the above example, as `%put` works at compilation time and would always write to the log. Something I like to do when developing or debugging code is to add a distinctive note to the log

```
%put cheese ;
```



Something easily searchable, and it helps if it makes you smile.

NOTES OF INTEREST

There are other things in the SAS log that can be useful to understand how your program has run, I will note some here.

RUNNING TIME

The log will tell you how long SAS took to run each proc and datastep. Both the time used in the CPU (central processing unit) and the actual "real" time. If a programming is running slowly you can check these times to see if a particular step is slowing things down and so where it is particularly worth making code more efficient.

```
NOTE: PROCEDURE GPLOT used (Total process time):
      real time           0.14 seconds
      cpu time            0.10 seconds
```

NUMBER OF RECORDS READ IN AND OUT

SAS notes in the log the number of records read in and out in each step.

```
data phuse16 ;
  set phuse1 ;
run ;
```

```
NOTE: There were 4 observations read from the data set WORK.PHUSE1.
NOTE: The data set WORK.PHUSE16 has 4 observations and 3 variables.
```

```
data phuse17 ;
  set phuse1 ;
  x=1 ;
  output ;
  x=2 ;
  output ;
run ;
```

```
NOTE: There were 4 observations read from the data set WORK.PHUSE1.
NOTE: The data set WORK.PHUSE17 has 8 observations and 4 variables.
```

This is useful for confirming that your steps are working as expected, in the first datastep above we read in and out the same number of records, in the second each record read in leads to 2 records read out.

ERROR FLAGS AND SUPPRESSING THEM

SAS also has an automatic variable _error_ so it can keep track of whether data is in the required format, this variable is in the PDV and is set to 0 when all is fine and 1 otherwise. If _error_ is 1 then SAS doesn't automatically throw an error but the records with issues are printed to the log.
It is useful to understand the detail of the PDV, however due to limited space I have included a reference in recommended reading.

ERROR FLAGS
In this example the year (a 4-digit number) conflicts with the expected date (DDMMMYYYY) values of the date9 informat.

```
data phuse2 ;
  set phuse1 ;
  new=input(year,date9.) ;
run ;
```

```
NOTE: Invalid argument to function INPUT at line 637 column 7.
year=2016 city=Barcelona country=Spain new=. _ERROR_=1 _N_=1
NOTE: Invalid argument to function INPUT at line 637 column 7.
year=2017 city=Edinburgh country=Scotland new=. _ERROR_=1 _N_=2
NOTE: Invalid argument to function INPUT at line 637 column 7.
year=2018 city=Frankfurt country=Germany new=. _ERROR_=1 _N_=3
NOTE: Invalid argument to function INPUT at line 637 column 7.
year=2019 city=Amsterdam country=TheNetherlands new=. _ERROR_=1 _N_=4
```

SAS cannot process these numbers with the date9 format. There is no error in this log, but there are clearly issues.

If you don't want your log full of notes about data which has not yet been cleaned, then you can set the error number lower, this also gives a warning as SAS has reached the allowed number of issues.

```
option errors=3 ;
```

```
NOTE: Invalid argument to function INPUT at line 632 column 7.
year=2016 city=Barcelona country=Spain new=. _ERROR_=1 _N_=1
NOTE: Invalid argument to function INPUT at line 632 column 7.
year=2017 city=Edinburgh country=Scotland new=. _ERROR_=1 _N_=2
NOTE: Invalid argument to function INPUT at line 632 column 7.
WARNING: Limit set by ERRORS= option reached.  Further errors of this type will not be printed.
year=2018 city=Frankfurt country=Germany new=. _ERROR_=1 _N_=3
```

CONCLUSION

The SAS log is very powerful in getting good reliable output when using SAS.
As we work in an industry where traceability and reuse of code are important, we need to take care in creating programs that are well commented and easy to follow. Good clean logs are an essential part of this.

REFERENCES

http://support.sas.com/documentation/cdl/en/lrcon/65287/HTML/default/viewer.htm#n1g8q3l1j2z1hjn1gj1hln0ci5gn.htm

RECOMMENDED READING

If you would like a deeper understanding of how the errors work, then learning about the PDV (program data vector) would be helpful.
http://support.sas.com/resources/papers/proceedings13/125-2013.pdf

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:
Sarah Berenbrinck
Independent
Email: sberenbrinck@gmail.com