Paper 350-2013

Best Practices: PUT More Errors and Warnings in My Log, Please!

Mary F. O. Rosenbloom, Edwards Lifesciences LLC, Irvine, CA

Kirk Paul Lafler, Software Intelligence Corporation, Spring Valley, California

ABSTRACT

We all like to see a SAS® log that is free from errors and warnings, but did you know that you can add your own errors and warnings to the log with PUT statements? Not only that, but you can incorporate this technique into your regular coding practice to check for unexpected data values. This paper will explore the rationale and process of issuing user-created error and warning messages to the SAS log, along with a number of examples to demonstrate when this is useful. Finally, we will propose an upgrade to the next version of SAS involving a user-specified keyword with its own color in the log.

INTRODUCTION

There are lots of steps that one goes through when running a SAS program in order to ensure that the program has run correctly. Most of us start by checking the log to ensure that it is free from ERROR and WARNING messages, as well as from other specific keywords. Next one might check the number of records in some of the data steps, or perhaps look at some of the output in the results window to view extreme values. It can be difficult to document this process, and some of these steps may get forgotten over time or missed when the program is run by someone else. One solution is to build all of the checks into your SAS program and to use PUT statements to send ERROR and WARNING messages to the log. These keywords are shaded (usually red for ERROR and green for WARNING), which adds emphasis to important messages.

This paper will illustrate the process for sending ERROR and WARNING messages to the log. We will discuss several applications of this technique, such as finding errors in the coding of variable values, identifying the location of the problem in the code, and flagging impossible data values. We will also discuss a technique to ensure that the terms "ERROR:" and "WARNING:" will not be found in the code unless they are issued by the system, or unless the PUT statements of interest are processed.

USING A PUT STATEMENT TO ADD ERRORS AND WARNINGS TO THE LOG

It's easy to add ERROR and WARNING statements to the SAS log. Many of us have used PUT within a data step to write a message to the log. Any time we type "WARNING:" with all capitals and the colon, anything else on that line in the log will be colored green. This can also be used to generate red text by typing "ERROR (Carpenter 2012, p.474, or Henderson).

The following example uses the built-in SASHELP.SHOES dataset ❶, which is organized by region, subsidiary, and product ❷. Suppose we want to determine whether or not there are duplicate products within a subsidiary. To do this, we identify any cases that are not simultaneously the first ❸ and last ❹ product within a subsidiary. We then use a PUT statement ❺ to send a message to the log to notify us. By typing "WARNING:" ❻ we ensure that this line in the SAS log will be shaded green. We can type any other message that we want to after the WARNING: statement. In this case we add the message "Multiple products in" and then follow it by a space. ❼ Next, we show the value of SUBSIDIARY ❽, followed by a slash, which causes a carriage return ❾. Everything after this line in the log will be black again unless it contains a 'WARNING:' or 'ERROR: ' keyword. In the next line of the log we also want to see the value of PRODUCT, but this time we add the equals sign. The log will now show 'PRODUCT=' before the value of product.❿

```
data shoes;
  set sashelp.shoes ❶;
  by region subsidiary product ❷;
  if first.product ❸ and not last.product ❹ then
     put ❺ 'WARNING: ❻ Multiple products in ' ❼ subsidiary ❽ / ❾ product= ❿;
run;
```

A portion of the SAS log is shown here:

```
WARNING: Multiple products in Copenhagen
Product=Sport Shoe
```

As expected, the WARNING statement appears in green and includes our custom message, complete with subsidiary name. The next line, in black, shows that the value of the variable PRODUCT is “Sport Shoe.”

IF, THEN, ELSE OR ELSE!

So now that we know the mechanics of how to add ERROR and WARNING statements to the SAS log, let’s look at another use for this technique. Many of us have had to work with “young databases” that have few observations. In fact, some datasets may not have any observations at all. If the data structure is not well documented, how can we ensure that the code that we are writing or validating is correct? We can use PUT statements to check for unexpected values in the data and to send a message to the log when they are detected.

```
data _null_;
  set visits ❶;
  if visit='1 Year' then do;
  /*<Code truncated>*/
  end;
  else
  if visit='2 Years' ❷ then do;
      /*<Code truncated>*/
  end;
  else
  if visit ne '' ❸ then put
               'WARNING: Unexpected value for visit' / visit=//;
run;
```

In the code illustrated above, a fictitious dataset, VISITS, is processed ❶. Perhaps none of the patients have reached their two-year follow-up visit by the time the SAS program is written. We want to write a program that can be used at any time during the study, but perhaps we don’t know the value that the variable VISIT will take at two years.

In this example the very last statement is a catchall. We guess that the value of VISIT at two years will be “2 Years” ❷, but we add a final clause ❸ so that if a non-missing value other than ‘1 Year’ or ‘2 Years’ appears, a warning will be sent to the log, and the user will be alerted.

```
WARNING: Unexpected value for visit
visit=2 Year
```

A partial log is shown above. In this case the value for the two year visit was “2 Year” rather than “2 Years”, and the WARNING statement in the log ensures that the user is aware of this.

PUT MORE ERRORS AND WARNINGS IN MY LOG, BUT ONLY WHEN NECESSARY!

In some companies, as part of standard operating procedures, it is never acceptable to have the terms “ERROR:” or “WARNING:” anywhere in the SAS log, even if they are just in a data step as part of a PUT statement that is not executed. Fear not! There is a work-around for this.

The next example shows how one might use PUT statements to identify impossible values in the data.

```
data bmi;
  set sashelp.class ❶;

  ***create an impossible value ❷;
  if _n_=3 then height=5;

  bmi=(weight/height**2)*703 ❸;

  ***check for impossible values of BMI ❹;
  if not missing(bmi) and bmi lt 1 or bmi gt 100 then
     put 'ERR' 'OR: ❺ BMI value is impossible.  Check calculation for '
     name / weight= height= bmi=/;
run;
```

Here we use the built-in dataset SASHELP.CLASS ❶, and we create an impossible value of five inches for HEIGHT ❷. Then we calculate BMI ❸ and use a PUT statement to check for non-missing BMI values less than 1 or greater than 100 ❹. We send an ERROR statement to the log ❺, since we want any impossible values to have the highest possible urgency in the log. So that we only have the word "ERROR:" in the log when there is truly an error, we break it into two words in the PUT statement. In many companies, an error in the log means that the program must be fixed or the data must be corrected before the program can be run successfully. In this case, one should use an ERROR statement sparingly and carefully.

A snapshot of the log is shown below. The error message shows up correctly even though the word was broken into two pieces in the data step.

```
ERROR: BMI value is impossible.  Check calculation for Barbara
Weight=98 Height=5 bmi=2755.76
```

Notice that the two words were separated by a space in the data step. Without the space, the pair of single quotes is resolved as one quote, and the log is not shaded.

```
ERR'OR: BMI value is impossible.  Check calculation for Barbara
Weight=98 Height=5 bmi=2755.76
```

Special thanks to Wei Cheng of Isis Pharmaceuticals for suggesting this technique.

LOCATING THE PROBLEM CODE

Sometimes we run several SAS programs in sequence, processing thousands of lines of code. When PUT statements from these programs are sent to the SAS log, it can sometimes be hard to determine which program activated the message. One might consider adding information to the PUT statement about which program, dataset, or step originated the message.

```
*****************************************************************************;
***STEP 5: Read in the CARS data and look for high MSRP to Invoice Ratios;
*****************************************************************************;
data cars;
  set sashelp.cars;
  if invoice gt 0 then ratio=msrp/invoice;
  if ratio gt 1.16 then put
        'WARNING: High Ratio of MSRP to Invoice, Step 5 of cars.sas program'/
        msrp=/ invoice=/;
run;
```

In this example, the code was broken up into steps. The program name, "cars.sas", and the step number are placed into the PUT statement. This allows us to quickly zero in on the section of code in the specified program, even if we had just run 40 programs. If the program is modified, the line numbers may change, but the step numbers are less likely to do so. The partial SAS log below illustrates the use of the PUT statement.

```
WARNING: High Ratio of MSRP to Invoice, Step 5 of cars.sas program
MSRP=$84,165
Invoice=$72,206
```

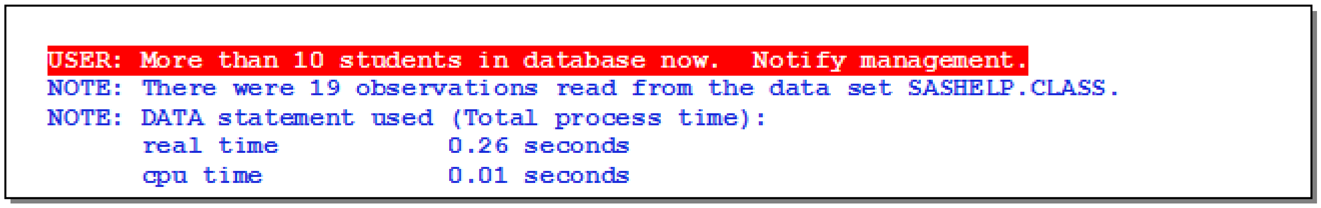
Special thanks to William E. Benjamin, Jr., of Owl Computer Consultancy, LLC for suggesting this application.

WOULDN'T IT BE NICE IF SAS COULD DO THIS?

So far we have seen that ERROR and WARNING statements can be added to the SAS log to emphasize program and data issues with colored messages. Wouldn't it be nice to have a keyword that could only be issued by the user and that got its own special shading in the log?

```
data _null_;
  set sashelp.class;
  if _n_=11 then
       put 'USER: More than 10 students in database now. Notify management.';
run;
```

In this example, the keyword 'USER:' shows up as white text with a red background in the log. As long as that keyword is never generated by SAS, the programmer could identify these types of messages easily and use them to keep track of issues with programs and data.



```
USER: More than 10 students in database now.  Notify management.
NOTE: There were 19 observations read from the data set SASHELP.CLASS.
NOTE: DATA statement used (Total process time):
      real time           0.26 seconds
      cpu time            0.01 seconds
```

This does not currently exist, but if it did, there would be lots of uses for it.

SUMMARY

Adding statements to the SAS log is an easy way to call attention to important issues. This technique is especially useful for programs that will be passed on to others to run, since we can't always assume that programmers will use what is printed in the results window to determine whether or not the program is running correctly. However, sending ERROR and WARNING messages should be done sparingly, and only to emphasize very critical issues, using WARNINGs to show important issues, and ERRORs to identify show-stoppers.

There are many reasons why you should put WARNINGs and ERRORs in your SAS log. We have discussed just a few here. Hopefully you will be inspired to adapt this to your own work to meet the needs of your programs, and hopefully future versions of SAS will permit a user-only keyword with its own color!

ACKNOWLEDGMENTS

The authors would like to thank Scott Leslie, Academic Program Chair, and Sally Carson, Operations Chair for a great conference. We would like to thank Ethan Miller and Mary McCracken, Co-Chairs for Coder's Corner, for their support. We would also like to thank William E. Benjamin, Jr., and Wei Cheng for their insights. Finally, Mary would like to thank Kirk for his enthusiastic collaboration and mentoring.

REFERENCES

Carpenter, Art (2012), Carpenter's Guide to Innovative SAS® Techniques, SAS Institute Inc., Cary, NC, USA.

Don Henderson explains how to use PUT statements to add color to the SAS log:
http://www.sascommunity.org/wiki/Tips:Using_NOTE,_WARNING,_ERROR_in_Your_Program's_Generated_Messages

Mengelbier, Magnus (2008), "Simple %str(ER)ROR Checking in Macros" SAS Institute, Inc. Proceedings of SAS Global Forum 2008. Cary, NC: SAS Institute Inc.

SAS documentation on the PUT statement:
http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/viewer.htm#a000161869.htm

Sezen, Burak and Sandeep Guadana (2005), "Error Handling: An Approach for a Robust Production Environment" SAS Institute, Inc. Proceedings of NESUG 18. Cary, NC: SAS Institute Inc.

Whitlock, Ian (2008), "The Art of Debugging" SAS Institute, Inc. Proceedings of SAS Global Forum 2008. Cary, NC: SAS Institute Inc.

TRADEMARK CITATIONS

ABOUT THE AUTHORS

Mary Rosenbloom is a statistical programmer at Edwards Lifesciences in Irvine, California. She has been using SAS for over 15 years, and is especially interested in using macros to generate data-driven code, DDE, and program validation methods.

Kirk Paul Lafler is consultant and founder of Software Intelligence Corporation and has been using SAS since 1979. He is a SAS Certified Professional, provider of IT consulting services, trainer to SAS users around the world, and sasCommunity.org emeritus Advisory Board member. As the author of four books including PROC SQL: Beyond the Basics Using SAS, Kirk has written more than five hundred papers and articles, been an Invited speaker and trainer at three hundred-plus SAS user group conferences and meetings, and is the recipient of 19 “Best” contributed paper, hands-on workshop (HOW), and poster awards.

Comments and suggestions can be sent to:

Mary F. O. Rosenbloom
Edwards Lifesciences, LLC
E-mail: mary.rosenbloom.sas@gmail.com
~~~
Kirk Paul Lafler
Software Intelligence Corporation
E-mail: KirkLafler@cs.com
LinkedIn: http://www.linkedin.com/in/KirkPaulLafler
Twitter: @sasNerd
~~~