The BEST. Message in the SASLOG®

Andrew T. Kuligowski, HSN

ABSTRACT

Your SAS® routine has completed. It is apparently a success – no bad return codes, no ERROR or WARNING messages in the SASLOG, and a nice thick report filled with what appear to be valid results. Then, you notice something at the end of the SASLOG that warns you that, somewhere in your output, one or more numbers are NOT being printed with the formats you requested.

```
NOTE: At least one W.D format was too small for the number to be
      printed.  The decimal may be shifted by the ""BEST"" format.
```

This presentation will review an ad hoc that can be used to quickly identify numbers that are too large for your selected format – certainly much quicker and effectively than attempting to eyeball a massive report!

THE PROBLEM

Most users of the SAS system have encountered the following message:

```
NOTE: At least one W.D format was too small for the number to be
      printed.  The decimal may be shifted by the "BEST" format.
```

You can refer to the manuals to learn the use and benefits of the **BEST.w** format, which is the default for numeric variables. However, most users of the SAS System prefer to embellish their output by using the various output formats available to them. The message listed above informs you that SAS encountered a minor problem with their routine, and is overriding its original instructions in order to complete its task without error. The message *implies*, however, that you do not understand your data as well as you might. In order to eliminate this message, you will want to isolate the values that are too large for their format, and modify the formats on your PUT statements, so that you no longer have this condition.

In many cases, the offending data is blatantly obvious on the SAS routine's output. A quick visual scan of the report will identify the number or numbers whose formats have been adjusted for printing. You can simply correct those formats and re-execute your SAS routine, without having to perform an extensive analysis or needing to write and execute assorted ad-hoc routines. In other situations, the problematic output is not as easy to spot. For example, the report could be very large, with most of the values within it conforming to the expected format. Or, it might be in a .CSV format, which is harder for the typical human to scan over than columnar reports. In these cases, you can use some basic tools that the SAS system provides to probe your data.

```
629  DATA  FORMAT42;
630    INFILE CARDS;
631    INPUT  @  1  ACTUAL $CHAR5.
632           @  1  FMT4_2      5.;
633    FILE  LOG ;
634    PUT @  1  ACTUAL= $CHAR5.
635        @ 15  FMT4_2=      4.2 ;
636  CARDS;
ACTUAL=7.499  FMT4_2=7.50
ACTUAL=14.49  FMT4_2=14.5
ACTUAL=768.1  FMT4_2=768
ACTUAL=1997   FMT4_2=1997
ACTUAL=4858.  FMT4_2=4858
ACTUAL=54632  FMT4_2=55E3
NOTE: The data set WORK.FORMAT42 has 6 observations and 2 variables.
NOTE: At least one W.D format was too small for the number to be printed. The
      decimal may be shifted by the "BEST" format.
NOTE: The DATA statement used 0.55 seconds.
```

Table "1" : Sample Data Illustrating "BEST." Format Override

Let us illustrate this situation with a very simple example – let's say that you will input a series of numbers using a numeric 5. format, and attempt to output them with a numeric 4.2 format. [**Table "1"** displays the routine used to read and write the values. It also contains a table showing the actual value input by the program matched against the value output via the BEST. format.]

THE ANALYSIS

You need to use a basic assumption to validate the numbers in this example : a 4.2 format should produce a single-digit number, followed by a decimal point and two decimal places. Therefore, the decimal place should always be in the same position -- the second from the left -- when the number is printed. The PUT function can be used to store the number to a character variable using this selected format. (It is suggested that you use the Zw.d format, which zero-pads the number to the left if necessary. This will ensure that the assumption of aligned decimals will be valid in examples when the number in question is expected to be greater than 9.) Once the formatted value is stored electronically, you can use the INDEX function to determine if the decimal place is in the expected position. [**Table "2"** contains the validation ad-hoc, along with a tabular listing of its results.]

```
644  DATA  _NULL_ ;
645    SET  FORMAT42;
646    C_FMT4_2 = PUT( FMT4_2,  Z4.2 );
647    WHERE_PT = INDEX( C_FMT4_2, '.' );
648    FILE  LOG ;
649    PUT @  1  ACTUAL    $CHAR5.
650        @  7  C_FMT4_2  $CHAR4.
651        @ 12  WHERE_PT       1.;
652  RUN ;

7.499 7.50 2
14.49 14.5 3
768.1 0768 4
 1997 1997 0
4858. 4858 0
54632 55E3 0
NOTE: At least one W.D format was too small for the number to be printed. The
      decimal may be shifted by the "BEST" format.
NOTE: The DATA statement used 0.28 seconds.
```

Table "2" : Validation Ad-Hoc (and Results) #1

The combined use of the PUT and INDEX functions can also be used to isolate data that exceed the anticipated precision when your values do not contain decimal places. When a whole number exceeds the expected precision, the BEST. format override will use scientific notation. You can use the INDEX function to locate the first occurrence of "E" in your number, as formatted by the PUT function. A value of 0 indicates that "E" is not present; this is the expected condition. However, a non-zero value can be interpreted to mean that SAS converted our number to scientific notation.

This discussion of which values of the INDEX function are and are not valid under certain circumstances can get confusing. You may find it easier to convert them into text, to clearly differentiate valid from invalid values. [**Table "3"** contains a complete example of this validation technique, illustrating the search for both invalid decimal places and scientific notation, and the conversion of numeric responses into text values.]

```
673  DATA  _NULL_ ;
674    SET  FORMAT42;
675    C_FMT4_2 = PUT( FMT4_2,  Z4.2 );
676    WHERE_PT = INDEX( C_FMT4_2, '.' );
677    WHERE_E  = INDEX( C_FMT4_2, 'E' );
678    IF  WHERE_PT ^= 2  THEN
679      ERRNOTE1 = 'DECIMAL';
680    IF  WHERE_E  ^= 0  THEN
681      ERRNOTE2 = 'EXPONENTIAL';
682    FILE  LOG ;
683    PUT @  1  ACTUAL    $CHAR5.
684        @  7  C_FMT4_2  $CHAR4.
685        @ 13  ERRNOTE1  $CHAR10.
686        @ 24  ERRNOTE2  $CHAR12.;
687  RUN ;

7.499 7.50
14.49 14.5  DECIMAL
768.1 0768  DECIMAL
 1997 1997  DECIMAL
4858. 4858  DECIMAL
54632 55E3  DECIMAL    EXPONENTIAL
NOTE: At least one W.D format was too small for the number to be printed. The
      decimal may be shifted by the "BEST" format.
NOTE: The DATA statement used 0.28 seconds.
```

Table "3" : Validation Ad-Hoc (and Results) #2

THE SOLUTION

In this example, you can see that the largest value being read in contains 5 significant digits to the left of the decimal place. You can also see that the most precise value contains 3 decimal places. There are several possible solutions to our problem. If you want to print all values that you have read in, you can change the code to output the value using a Z9.3 format, or you can change the code to output the value using a Z8.2 format if you only need two decimal places. On the other hand, if you want to flag large numbers as erroneous values, you can put in validation code that watches for and traps numbers >=10, only printing values that can be properly printed in the requested 4.2 format.

CONCLUSION

This paper addressed one message that is commonly found in a SASLOG. It discussed the use of an ad hoc routine to explore WHY that message occurred, and covered how to modify a routine to prevent the recurrence of that message. It is hoped that the mechanisms discussed in this paper might be used by the readers in their daily jobs. However, this paper is a failure -- at least in part -- if the process stops there. It is hoped, even more strongly, that the *concepts* of developing and using ad hoc routines to fully understand ones data are the *true* lessons that the reader retains from this paper.

BIBLIOGRAPHY
REFERENCES / FOR FURTHER INFORMATION

Burlew, Michele M. (2001). *Debugging SAS Programs – A Handbook of Tools and Techniques*. Cary, NC: SAS Institute, Inc.

Kuligowski, Andrew T. (2002), "Pruning the SASLOG – Digging into the Roots of NOTEs, WARNINGs, and ERRORs". *Proceedings of the New Mexico SAS Users Day Conference of the SouthCentral SAS Users Group*. USA.

SAS Institute, Inc. (1990), *SAS Language: Reference, Version 6, First Edition.* Cary, NC: SAS Institute, Inc.

SAS Institute, Inc. (1994), *SAS Software: Abridged Reference, Version 6, First Edition.* Cary, NC: SAS Institute, Inc.

The author can be contacted via e-mail at:
KuligowskiConference@gmail.com