Software Validation and Testing

Andrew T. Kuligowski, Nielsen Media Research

ABSTRACT

You've sat through constant design meetings. You've endured countless requests for "just one more little change". You even managed to find a creative solution to that nagging technical problem. But, you persevered, and despite all of the obstacles, you've managed to eliminate the final syntax error in your newest SAS® routine. Time to sit back and relax -- uh, not quite ...

This tutorial will address the issues of testing your routines and validating your data. The primary focus will be on techniques to ensure comprehension of your input data - SASLOG messages such as

```
NOTE: MERGE statement has more than one data set with repeats of
BY values.
```

imply that there may be gaps in your knowledge of your data! Special emphasis will be placed on the use of ad-hoc queries to assist in finding data anomalies that can cause problems with your SAS routine.

INTRODUCTION

"Software Validation and Testing" is an important topic that is often given insufficient attention. In an ideal world, analysts and software engineers would be assembling a Test Plan at the same time they are working on the initial System Design. The Test Plan would include a reference to unit testing each individual module and SAS routine in the system, although that "reference" would probably be a cross-reference to a series of separate documents -- one per routine. It would include a comprehensive System Test Plan to review how those components work together towards the common goal of the entire system, and it would cover an Acceptance Test plan for the end user to verify that the completed system performed according to expectations.

This paper would perform an injustice in attempting to compress those topics into the limited space and time allotted for this presentation. Therefore, this paper will emphasize one aspect of software testing -- comprehension of your input data. We will address the validation of the interface between a SAS routine and its input data, with emphasis on what to do once you realize that you have an incomplete understanding of that data.

The SAS messages described in this paper are output to the SAS log during program execution. It is assumed that the reader has a basic understanding of the SAS log, including its composition, format, and the SAS system options which control its content.

MERGE Statement: Repeats of BY Values

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

```
NOTE: MERGE statement has more than one data set with repeats of BY values.
```

There are many papers that can be found in the Proceedings from past SUGI and the various regional SAS User Group conferences that describe how to programatically force such a merge to occur. This assumes that the user *wants* to merge datasets that have repeats of BY values. However, it is also possible that the user did not expect this condition. This implies an error in your SAS routine, caused by a misunderstanding of the input data. We want to isolate these cases and modify our assumptions, so that we can correct our MERGE and eliminate this condition.

The following will illustrate an example of "repeats of BY values". We are going to merge a dataset containing a list of dog breeds [see Table "1-A"] against a dataset containing the dogs owned by sample

households [see Table "1-B"] using the variable BREED, keeping only those records from the Breed file that have a corresponding record in the Household file. As one would expect (knowing, of course, that this example was created to illustrate the problem at hand), the merge results in "more than one data set with repeats of BY values" [illustrated in Table "1-C"]. The problem is to determine which BY values had repeats, which records (on which files) are affected, and what additional information needs to be included to make the "BY values", also referred to as "merge variables", unique on at least one of the files.

BREED	VARIETY	OTHER INFO
Bulldog		17834
Dalmatian		49235
Dachshund	Mini	18435
Dachshund	MiniLonghair	75846
Dachshund	MiniWirehair	09431
Dachshund	Std	18098
Dachshund	Std Longhair	75324
Dachshund	Std Wirehair	09389
Ger Sheprd		09622
GoldRetrvr		38292
Husky,Sib		75555
Lab Retrvr		38192

Table "1-A" : Breed File

HHLD_ID	BREED	VARIETY	BIRTHDAY	GOTCHADY	GOAWAYDY
0005884	Dalmatian		07/31/87	10/14/87	
0005884	Dalmatian		12/23/89	12/23/89	
0005884	Bulldog	English	05/19/91	05/19/94	02/20/95
0005884	Dachshund	Std Longhair	09/17/94	03/28/95	
0005884	Dachshund	Std Longhair	10/29/95	01/31/96	
0008824	Ger Sheprd		11/24/89	01/01/90	12/07/92
0008824	Husky,Sib		05/26/93	07/20/93	
0008824	Lab Retrvr		02/28/94	05/07/94	05/06/95
0008824	GoldRetrvr		03/06/95	05/06/95	

Table "1-B" : Dogs per Household File

```
177   DATA DOGDTAIL;
178      MERGE  DOGOWNED (IN=IN_OWNED)
179             DOGBREED (IN=IN_BREED);
180      BY  BREED ;
181      IF  IN_OWNED ;
182   RUN;

NOTE: MERGE statement has more than one data set with repeats
      of BY values.
NOTE: The data set WORK.DOGDTAIL has 13 observations and
      7 variables.
```

Table "1-C" : MERGE with "repeats of BY values"

We can use basic elements of the SAS System to do most of the analysis for us; our most powerful tool will be the MEANS procedure. PROC MEANS is traditionally used to compute descriptive statistics for numeric variables in a SAS Data set. In this situation, we simply need a list of the unique values of our merge variable (or, in a more complex case, the unique combinations of values for our merge variables), along with a count of the number of occurrences of each in both of our datasets. The NWAY option on the PROC MEANS statement will limit the output dataset to only those observations with the "highest interaction among CLASS variables" -- that is, only those records containing the unique values or combinations of values for the BY variables will be written to the output dataset. (NOPRINT is optional; it can be used or excluded based on personal preference.) The variable used in the VAR statement can be any numeric variable in the dataset; the only condition is that it must be a *numeric* variable. Date variables and ID values should be

considered, since many / most datasets contain them and they are normally stored as numeric values. (In the rare event your dataset contains exclusively character variables, you will need to add a numeric variable to either the original dataset or a copy of it prior to issuing PROC MEANS.) The OUTPUT statement must specify an OUT= dataset. It must also include the statistics keyword N=, so that a record count -- and only a record count -- for each unique combination of values for the BY variables will be written to each observation of the output dataset. [Table "1-D" illustrates this use of PROC MEANS for one of our two example datasets; the code for the other dataset in our example is almost identical.]

```
505   PROC MEANS DATA=DOGOWNED  NOPRINT  NWAY;
506       CLASS  BREED ;
507       VAR HHLD_ID ;
508       OUTPUT  OUT=SUMOWNED  N=;
509   RUN;

NOTE: The data set WORK.SUMOWNED has 7 observations and
      4 variables.
```

Table "1-D" : PROC MEANS example

The next step is to take the outputs of our PROC MEANS for each affected dataset and merge *them* together. We *cannot* encounter the "... repeats of BY values" note, since we now only have one observation per unique combination of BY values! Each observation *does* have a count of the number of observations that contain the combination of BY values in their original dataset, stored as _FREQ_. The RENAME= option the datasets in the MERGE statement can be used to give the _FREQ_ variable in each dataset a unique name in our output dataset. We will use a subsetting IF, so that we only keep those observations that have multiple occurrences in each input dataset. The output of this step will contain the unique combinations of values that are causing the "...repeats of BY values" note. [The routine is contained in Table "1-E", and the output depicted in Table "1-F".]

```
585   DATA SUMMERGE (KEEP=BREED  CNTBREED  CNTOWNED);
586       MERGE  SUMBREED (RENAME=(_FREQ_=CNTBREED))
587              SUMOWNED (RENAME=(_FREQ_=CNTOWNED));
588       BY  BREED ;
589       IF  CNTBREED > 1  AND  CNTOWNED > 1;
590   RUN ;

NOTE: The data set WORK.SUMMERGE has 1 observations and
      3 variables.
```

Table "1-E" : Merging the PROC MEANS outputs

SAS Dataset WORK.SUMMERGE

OBS	BREED	CNTBREED	CNTOWNED
1	Dachshund	6	2

Table "1-F" : Results of Merging the PROC MEANS outputs

Up to this point, we have not discussed one important factor in this anelysis - the human element. The process described in this section is meant to be used as a tool to guide the analyst through the unknown elements in their data - once these areas become *known* - there is no need to continue this analysis. The listing of unique combinations of values that occur multiple times in each dataset will often be that stopping point for an analyst. The information obtained will allow them to make modifications to their assumptions and corresponding changes to their routines. However, in the event that the oddities are still not clear, we can employ one or more additional MERGE steps, taking the merged outputs from the PROC MEANS, and merging *that* dataset against each of the original datasets. [Table "1-G" shows how this is done for one of our two input datasets; the code for the other is almost identical]. This final step should provide sufficient clarification for the analyst to determine which factor(s) are missing in their assumptions and adjust their routines accordingly.

```
599   DATA  CHKOWNED ;
600      MERGE  DOGOWNED (IN=IN_BREED )
601             SUMMERGE (IN=IN_MERGE );
602      BY  BREED ;
603      IF  IN_MERGE ;
604   RUN ;

NOTE: The data set WORK.CHKOWNED has 2 observations and
      8 variables.
```

Table "1-G" : Merging the analysis back to the original input

INPUT Statement: Reached past the end of a line

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

```
NOTE: SAS went to a new line when INPUT statement reached past
      the end of a line.
```

The manuals describe how to prevent this message using options on the INFILE statement. MISSOVER and TRUNCOVER will prevent SAS from reading the next line but continue processing, while STOPOVER will force an error condition and stop building the data set. However, these options do not help resolve which line(s) on the input dataset triggered the problem. This message implies an error in your SAS routine, caused by a misunderstanding of the input data. We want to isolate these cases and modify our INPUT statement so that we no longer have this condition.

We can illustrate this with an example. We will read a sequential file containing the ID numbers of students in attendance at classes per day. Each record will have a key containing the Date and Class Name. The next two fields will represent the Number of Students Registered and the Number of Students Absent. Finally, the record will have a variable number of Student IDs, representing the students in attendance at the class on the given date [shown in Table "2-A"]. Our first attempt at reading this file will use the Number of Students Registered field to determine - incorrectly - how many occurrences of the Student ID field must be processed. [Table "2-B" will illustrate the SAS default of attempting to complete the INPUT statement on the next line. Note that the SASLOG indicates that we have 3 observations, although we read a file with 5 lines.] The problem is to isolate the incorrect assumption in our original analysis that is causing us to continue the INPUT after we have hit the end of line.

DATE	CLASS	REGIST.	ABSENT	ID # OF STUDENTS IN ATTENDANCE
02/21/96	Physics	12	2	27 29 33 34 37 41 42 43 44 45
02/21/96	Botany	15	7	6 7 9 28 35 36 40 51
02/21/96	Geology	16	9	13 29 30 31 39 45 46
02/21/96	Anatomy	8	1	10 12 22 25 32 47 49
02/21/96	Zoology	10	0	1 3 7 8 9 12 18 19 22 23

Table "2-A" : Attendance File

```
15    DATA  ATTEND;
16       ARRAY ATNDID (25) ATNDID01-ATNDID25 ;
17       INFILE  'C:\SUGI21\TSTS-SAS\ATTEND.DAT';
18       INPUT  @  1  DATE     MMDDYY8.
19              @ 10  CLASS     $CHAR8.
20              @ 18  REGIST         2.
21              @ 21  ABSENT         2. @ ;
22       pt = 24 ;
23       DO CNT = 1 TO REGIST;
24          INPUT @ pt ATNDID(CNT) 2. @ ;
25          pt = pt + 3 ;
26       END ;
27    RUN;

NOTE: The infile 'C:\SUGI21\TSTS-SAS\ATTEND.DAT' is:
      FILENAME=C:\SUGI21\TSTS-SAS\ATTEND.DAT,
      RECFM=V,LRECL=256
NOTE: 5 records were read from the infile
      'C:\SUGI21\TSTS-SAS\ATTEND.DAT'.
      The minimum record length was 43.
      The maximum record length was 52.
NOTE: SAS went to a new line when INPUT statement reached
      past the end of a line.
NOTE: The data set WORK.ATTEND has 3 observations and
      31 variables.
NOTE: The DATA statement used 0.98 seconds.
```

Table "2-B" : Incorrect INPUT routine

As in the MERGE problem discussed earlier in this paper, we can use basic elements of the SAS System to do most of the analysis for us. In this case, we will use the options available on the INFILE statement itself. The LENGTH= option on the INFILE statement will define a numeric variable, which will be assigned the length of the current input line when an INPUT statement is executed. In our example, we will begin with the assumption that our initial input statement is correct and it is the array processing that is incorrect. Therefore, we will subtract the 22 bytes contained in our record's key from our line size. (Please note that the LENGTH= variable is not written to the output dataset; we will store its value in another SAS variable so that we can make further use of it in subsequent steps if necessary.) Finally, we will assume that our array is correctly made up of 2 character numeric variables, delimited by a single blank character. Therefore, we will divide the remaining line size by 3 to determine the number of members in the array per line; we will add 1 to the difference under the assumption that the final numeric value is not blank-padded. [Table "2-C" contains the SAS routine described in this paragraph, while Table "2-D" contains the dataset created by executing the routine.]

```
521   DATA  LINELONG;
522      INFILE  'C:\SUGI21\TSTS-SAS\ATTEND.DAT'
523              MISSOVER  LENGTH=LNSZ;
524      INPUT  @  1  DATE      MMDDYY8.
525             @ 10  CLASS      $CHAR8.
526             @ 18  REGISTCT        2.
527             @ 21  ABSENTCT        2. @ ;
528      LINESIZE = LNSZ;
529      STDNTCNT = ( LINESIZE - 23 + 1 ) / 3;
530   RUN;
```

Table "2-C" : Using LENGTH= to determine line size

DATE	CLASS	REGIST.	ABSENT	LINESIZE	STDNTCNT
02/21/96	Physics	12	2	52	10
02/21/96	Botany	15	7	46	8
02/21/96	Geology	16	9	43	7
02/21/96	Anatomy	8	1	46	8
02/21/96	Zoology	10	0	52	10

Table "2-D" : Results of our LENGTH= experiment

The final task is the trickiest one -- interpretation of the output. By quickly scanning the output, we can see that only one record has the same value, 10, for "REGIST" as for "STDNTCNT". The record has one other oddity -- the value of "ABSENT" is 0. As we all learned early on in life, 10 - 0 = 10. Therefore, we can adopt the working theory that STDNTCNT = REGIST - ABSENT. A quick check of our input data will show that the theory holds for every record in our dataset. (We could have written another ad-hoc routine to verify this assumption if our sample data had been more complex.) Therefore, by replacing the upper bound of our FOR loop with the calculated value STDNTCNT, our data can be read without error [as shown in Table "2-E"].

```
579   DATA  ATTEND;
580      ARRAY ATNDID (25) ATNDID01-ATNDID25 ;
581      INFILE  'C:\SUGI21\TSTs-SAS\ATTEND.DAT';
582      INPUT  @  1  DATE    MMDDYY8.
583             @ 10  CLASS   $CHAR8.
584             @ 18  REGIST       2.
585             @ 21  ABSENT       2. @ ;
586      STDNTCNT = REGIST - ABSENT ;
587      pt = 24 ;
588      DO CNT = 1 TO STDNTCNT ;
589         INPUT @ pt ATNDID(CNT) 2. @ ;
590         pt = pt + 3 ;
591      END ;
592   RUN;

NOTE: The infile 'C:\SUGI21\TSTS-SAS\ATTEND.DAT' is:
      FILENAME=C:\SUGI21\TSTS-SAS\ATTEND.DAT,
      RECFM=V,LRECL=256
NOTE: 5 records were read from the infile
      'C:\SUGI21\TSTS-SAS\ATTEND.DAT'.
      The minimum record length was 43.
      The maximum record length was 52.
NOTE: The data set WORK.ATTEND has 5 observations and
      32 variables.
NOTE: The DATA statement used 0.7 seconds.
```

Table "2-E" : Correct INPUT routine

In the real world, the solution may not be as apparent as it was for our contrived example. In these cases, it may be necessary to review the assumptions made at the start of the analysis. The investigation may have cast a doubt on their validity, or possibly even disproved some of them altogether. The analyst should adjust the assumptions that are believed to be incorrect and return to exploring their data with a fresh angle. It is also possible that the investigation has not been fruitful, but none of the working assumptions have been neither proved nor disproved. The analyst has two choices at this point. One possibility is to return to the analysis using an alternate tool. [For example, during the research period for this paper, the author explored the use of the $VARYING. informat as a mechanism to solve the problem. It proved to be less effective than the LENGTH= option described above, and was omitted from the final draft in the interests of space.] The other approach is to return to the analyst's assumptions and alter one or more of them. The analyst can then readdress the problem from a new angle -- the worst case will be that it proves no more fertile than the unproductive approach that the analyst had just abandoned!

BEST. Format

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.
```

The manuals describe 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 the user 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 the user does not understand their data as well as they might. We want to isolate the values that are too large for their format, and modify the formats on our PUT statements so that we 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. The analyst can simply correct their formats and re-execute their 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. In these cases, we can once again use the tools that the SAS system provides to probe our data.

We will illustrate this situation with a very simple example; we will input a series of numbers using a numeric 5. format and attempt to output them with a numeric 4.2 format. [Table "3-A" 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.]

```
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 "3-A" : Sample Data Illustrating "BEST." Format Override

We will use a basic assumption to validate our numbers : 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 our selected format. (It is suggested that the analyst 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, we can use the INDEX function to determine if the decimal place is in the expected position. [Table "3-B" contains our validation ad-hoc, along with a tabular listing of its results. The text contained in the table describes any problem(s) that might be present in the formatted value.]

```
645   DATA _NULL_ ;
646      SET  FORMAT42;
647      C_FMT4_2 = PUT( FMT4_2,  Z4.2 );
648      WHERE_PT = INDEX( C_FMT4_2, '.' );
649      WHERE_E  = INDEX( C_FMT4_2, 'E' );
650      IF  WHERE_PT ^= 2  THEN  ERRNOTE1 = 'DECIMAL';
651      IF  WHERE_E  ^= 0  THEN  ERRNOTE2 = 'EXPONENTIAL';
652      FILE  LOG ;
653      PUT @  1  ACTUAL     $CHAR5.
654          @  7  C_FMT4_2   $CHAR4.
655          @ 13  ERRNOTE1   $CHAR10.
656          @ 24  ERRNOTE2   $CHAR12.;
657   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-B" : Ad-Hoc (and Results) to Validate Output Format

The combined use of the PUT and INDEX functions can also be used to isolate data which exceed the anticipated precision when our values do not contain decimal places. When a whole number exceeds the expected precision, the BEST. format override will use scientific notation. We can use the INDEX function to locate the first occurrence of "E" in our 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. [In the interest of space, Table "3-B", displayed above, also contains an example of this validation technique.]

CONCLUSION

This paper addressed different messages that are commonly found in a SASLOG. It discussed the use of ad hoc routines to explore WHY those messages occurred, and covered how to correct a routine to prevent the recurrence of those messages. 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.

REFERENCES / FOR FURTHER INFORMATION

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:
0005949476@mcimail.com or
kuligoat@tvratings.com

ACKNOWLEDGMENTS

The author graciously offers thanks to Frank Dilorio, Neil Howard, David Riba, and Nancy Roberts for their contributions towards this paper. If they hadn't offered their suggestions on the content of this paper, agreed to proofread its myriad drafts, or offered the occasional reminder as to the definition of the word *deadline*, this paper would not have been completed. The author also wishes to acknowledge those individuals, whose identities have been lost to the ages, whose data oddities across the years provided the true inspiration behind this paper.