Paper CC15-2011

In Search of the LOST CARD

Andrew T. Kuligowski

ABSTRACT

"Everyone who's not here, raise your hand." It's an old joke, but it points out the difficulty of identifying persons or things that are not present. The SAS® System has its own version of this chestnut, the SASLOG message indicating that there are one or more gaps in one's input data:

```
NOTE: LOST CARD.
```

This presentation will focus on the creation and use of ad hocs to explore input data, in order to locate the positions where the input data might be incomplete. The goal will be to identify where the missing data should be, so that you can code around the limitations of your data.

INTRODUCTION

Anyone who has ever attempted to read an external file into SAS will most likely have encountered the **LOST CARD** note in their SASLOG. So, what exactly *is* a **LOST CARD**? The message indicates that SAS has encountered the End of File marker on the external file that was specified for input without having been able to read all the variables specified on the current INPUT statement. The wording of the note dates back to the early days of data processing, when punch cards ruled the computing world. In other words, the message could be taken literally - one or more cards must have slipped out of the deck prior to processing!

Nowadays, the message might actually indicate that the input dataset is missing one or more lines of data – assuming that your DATA step is coded to read multiple input lines per pass. It might also mean that SAS is attempting to read multiple lines of input data to create a single observation – which the author of the routine may not have intended – and it slammed into the End of File before it could complete its current observation.

The SASLOG will contain a clue as to which situation is most likely to be occurring, as described below. Knowing that all good SAS users religiously check their SASLOG (true?), the diligent analyst will encounter clues as to the nature of the problem.

TWO NOTES: LOST CARD with INPUT Statement reached past the end of a line

The first case we will examine is when **LOST CARD** is accompanied by another message:

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

This combination of messages is displayed when the INPUT statement is attempting to read more information than is present on a line of data. Perhaps the routine has the wrong input format, and is trying to read X bytes of data from a fixed-length file with a record length that is less than X bytes long. Alternatively, the routine could be processing a variable length file, and incorrectly determining the length of one or more lines.

One quick way to resolve this issue is to add some appropriate options to the INFILE statement. The manual describes two possible scenarios. The coder can use the MISSOVER and TRUNCOVER options to prevent SAS from reading past the end of the current line and continue processing normally. This approach does not explain why the INPUT statement(s) are not in sync with the actual file format. It does prevent SAS from performing its default behavior, which is attempting to complete its INPUT by grabbing data from the next line at the cost of setting the

unread variables to missing values. Plus, it allows the routine to continue without setting the ERROR flag. This combination of behaviors may or may not be acceptable to the coder. It should be noted, however, that this solution only masks the author's lack of knowledge of their input data – it does not explain and assist in correcting those discrepancies.

Alternatively, the user can incorporate the STOPOVER option to force an error condition and stop processing. This solution does not mask the problem and it prevents the routine from a successful conclusion. However, it does not actually resolve the situation; the coder must still research the situation and most likely make alterations to their routine. It should be noted that this approach avoids the possibility of a problem being missed during a routine scan of the SASLOG and should be considered as a fail-safe method.

This condition in question can best be illustrated with an example. In this case, the input file contains class attendance records, one record for each class for each date.

```
  Date    Class    Regist. Absent  Student IDs of attendees
02/21/05 Physics    12       2    27 29 33 34 37 41 42 43 44 45
02/21/05 Botany     15       7     6  7  9 28 35 36 40 51
02/21/05 Geology    16       9    13 29 30 31 39 45 46
02/21/05 Anatomy     8       1    10 12 22 25 32 47 49
02/21/05 Zoology    10       0     1  3  7  8  9 12 18 19 22 23
02/21/05 Chemistry   9       3    10 12 22 25 32 47
```

The SAS routine that reads this dataset is set up to read each line of data. The first four variables each have one occurrence and are in fixed columns at the beginning of each record. The Student IDs of the attendees are variable in number. In order to process this varying quantity of student IDs, the routine contains a loop based on the number of students registered.

```
3           DATA  ATTEND;
4              ARRAY ATNDID (25) ATNDID01-ATNDID25 ;
5              INFILE  'NEXTLINE DATA A';
6              INPUT  @  1  DATE    MMDDYY8.
7                     @ 10  CLASS    $CHAR8.
8                     @ 20  REGIST        2.
9                     @ 28  ABSENT        2. @ ;
10             pt = 34 ;
11             DO CNT = 1 TO REGIST;
12                INPUT @ pt ATNDID(CNT) 2. @ ;
13                pt = pt + 3 ;
14             END ;
15          RUN;
NOTE: The infile 'NEXTLINE DATA A' is:
      File Name=NEXTLINE DATA A,
      Lrecl=62,Recfm=V
NOTE: LOST CARD.
ATNDID01=10 ATNDID02=12 ATNDID03=22 ATNDID04=25 ATNDID05=32
ATNDID06=47 ATNDID07=. ATNDID08=. ATNDID09=. ATNDID10=. ATNDID11=.
ATNDID12=. ATNDID13=. ATNDID14=. ATNDID15=. ATNDID16=. ATNDID17=.
ATNDID18=. ATNDID19=. ATNDID20=. ATNDID21=. ATNDID22=. ATNDID23=.
ATNDID24=. ATNDID25=. DATE=16488 CLASS=Chemistr REGIST=9 ABSENT=3
pt=52 CNT=7 _ERROR_=1 _N_=3
NOTE: 6 records were read from the infile 'NEXTLINE DATA
A'.
      The minimum record length was 50.
      The maximum record length was 62.
NOTE: SAS went to a new line when INPUT statement reached past the
      end of a line.
NOTE: The data set WORK.ATTEND has 2 observations and 31
      variables.
```

Comment #2

Comment #1a

Comment #1b

As mentioned, this routine has some sort of logic flaw, causing both offending messages to be displayed in the SASLOG. In addition to the two NOTEs that were referenced earlier, there are other clues that this routine might not be doing what is expected or desired. (These comments can be cross-referenced with “Comment 1” and “Comment 2” as labeled in the above example.)

1) One NOTE states that the routine read 6 records, which matches the record count of the sample input file. A subsequent NOTE indicates that only 2 observations were written to the output SASDATA file. Simple arithmetic shows that we have somehow lost over 50% of our input data!
2) The LOST CARD note triggers a dump of the current observation to be displayed in the SASLOG. A quick examination of the variables listed shows that _N_=3. Since it has already been established that the routine only kept 2 observations, it can easily be concluded that the incomplete data from this last observation has been discarded. (As if already knowing that the routine has moved from 6 to 2 observations isn’t enough evidence of a problem!)

There are times where a quick examination of the routine, in conjunction with some sample input data, might provide a quick explanation of the problem; in the process, this will lend itself to a solution. (Given the simplistic nature of the example in this presentation, many readers may already see the logic flaw.) In other cases, it may be desirable to write some ad hoc queries to explore the input data and obtain more information about it.

The SAS System can be used to perform most of the analysis for us. In this case, the options available on the INFILE statement itself will be employed to explore the data. 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. The initial ad hoc will begin with the assumption that the initial INPUT statement is correct as coded, but the array / loop processing is incorrect. Therefore, the ad hoc will be coded to account for 30 bytes of “key” values. (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.) The other critical assumption for this first pass is that the array is correctly made up of 2 character numeric variables, delimited by a single blank character. Therefore, the routine will be set up to 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.

```
21   DATA  LINELONG;
22      INFILE  INFILE  'NEXTLINE DATA A'
23               MISSOVER  LENGTH=LNSZ;
24              INPUT  @  1  DATE    MMDDYY8.
25                     @ 10  CLASS    $CHAR8.
26                     @ 20  REGIST        2.
27                     @ 28  ABSENT        2. @ ;
28      LINESIZE = LNSZ;
29      STDNTCNT = ( LINESIZE - 33 + 1 ) / 3;
30   RUN;
```

DATE	CLASS	REGIST.	ABSENT	LINESIZE	STDNTCNT
02/21/96	Physics	12	2	62	10
02/21/96	Botany	15	7	56	8
02/21/96	Geology	16	9	53	7
02/21/96	Anatomy	8	1	53	7
02/21/96	Zoology	10	0	62	10
02/21/05	Chemistry	9	3	50	6

The results of the ad hoc are meaningless at first glance; they are meant to be tools for the analyst for further scrutiny. Therefore, the final task – the trickiest one – is the interpretation of

the output. A quick scan of the output reveals that only one record has the same value – 10 – for REGIST as for STDNTCNT; all other values of REGIST are smaller than STDNTCNT. Since REGIST is the value that was used to determine the number of array elements to be read and populated, it can easily be seen that the INPUT statement will overshoot the end of line almost every time.

The record has one other oddity -- the value of "ABSENT" is 0; all other observations show a positive non-zero integer. Going back to elementary school arithmetic, 10 - 0 = 10. Therefore, it appears that the code ignored the fact that some students may be absent from a given lecture. This provides a working theory that STUDENT COUNT = REGISTERED *minus ABSENT* is more accurate than what was originally implied in the code; STUDENT COUNT = REGISTERED. A quick check of the input data will show that the theory holds for every record in the dataset. (Another ad-hoc routine could have been coded and executed to verify this assumption if the sample data had been more complex.) Therefore, by replacing the upper bound of the DO loop with the calculated value STDNTCNT, the data can be read without error.

```
44          DATA  ATTEND;
45             ARRAY ATNDID (25) ATNDID01-ATNDID25 ;
46             INFILE  'NEXTLINE DATA A';
47             INPUT  @  1  DATE    MMDDYY8.
48                    @ 10  CLASS    $CHAR8.
49                    @ 20  REGIST        2.
50                    @ 28  ABSENT        2. @ ;
51             STDNTCNT = REGIST - ABSENT ;
52             pt = 34 ;
53             DO CNT = 1 TO STDNTCNT;
54                INPUT @ pt ATNDID(CNT) 2. @ ;
55                pt = pt + 3 ;
56             END ;
57          RUN;

NOTE: The infile 'NEXTLINE DATA A' is:
      File Name=NEXTLINE DATA A,
      Lrecl=62,Recfm=V
NOTE: 6 records were read from the infile 'NEXTLINE DATA A'.
      The minimum record length was 50.
      The maximum record length was 62.
NOTE: The data set WORK.ATTEND has 6 observations and 31
      variables.
```

In the real world, the solution may not be as apparent as it was in this contrived example. 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!

LOST CARD ALONE

The "end of line" notice does not always accompany the **LOST CARD** note. It is possible for the message to be displayed independently. This occurs when a DATA step has multiple INPUT statements, or uses a line pointer such as "/" to read multiple input lines with the same INPUT statement.

In the following example, an external dataset contains 3 separate lines that will be combined by the DATA step into a single observation in a SAS dataset. The first line contains the Name, Birth, and Death Date of a famous individual. The second line contains a text field containing a great accomplishment by that person, and the third line contains a second accomplishment. Each line is appropriately numbered 1, 2, or 3.

```
LINE 1 Franklin, Benjamin                1706 1790
LINE 2 First US Ambassador to France
LINE 3 Invented Lightning Rod
LINE 1 Truman, Harry S                   1884 1972
LINE 2 33rd President of United States
LINE 3 Ordered US Troops to Korea - 1950
LINE 1 Plante, Jacques                   1929 1986
LINE 2 First goaltender to regularly wear a mask
LINE 3 Won Hart Memorial Trophy (NHL MVP) 1961-62
LINE 1 Goodnight, James                  1943
LINE 2 Co-founder, SAS Inc. (nee SAS Institute)
LINE 3 Perennially listed on Forbes list of 100 richest people
LINE 1 Berry, Halle                      1966
LINE 2 2001 Academy Award Winner - Monster's Ball
LINE 3 1999 - Executive Producer - Introducing Dorothy Dandridge
LINE 1 Copernicus, Nicolaus              1473 1543
LINE 3 Theorized Earth not the center of universe
```

```
120         DATA  FAMOUS;
121            INFILE  'NEXTLINE DATA2 A';
122            INPUT   @  6  LINENUMA          1.
123                    @  8  NAME         $CHAR32.
124                    @ 41  BORN              4.
125                    @ 46  DIED              4.
126                  / @  6  LINENUMB          1.
127                    @  8  ACCOMPLISH1 $CHAR66.
128                  / @  6  LINENUMC          1.
129                    @  8  ACCOMPLISH2 $CHAR66. ;
130         RUN;
NOTE: The infile 'NEXTLINE DATA2 A' is:
      File Name=NEXTLINE DATA2 A,
      Lrecl=80,Recfm=FB,Blksize=960
NOTE: LOST CARD.
LINENUMA=1 NAME=Copernicus, Nicolaus BORN=1473 DIED=1543
ACCOMPLISH1=Theorized Earth not the center of universe
LINENUMB=3 LINENUMC=. ACCOMPLISH2=  _ERROR_=1 _N_=6
NOTE: 17 records were read from the infile
      'NEXTLINE DATA2 A'.
NOTE: The data set WORK.FAMOUS has 5 observations and
      8 variables.
```

Unfortunately, the SASLOG shows that the INPUT statement did not function as intended, since the infamous **LOST CARD** message is displayed.

Once again, the analyst should only perform the minimum amount of investigation necessary to resolve the problem at hand. Given the simplistic nature of the example and the minimal amount of test data, the erroneous record(s) may be easily identified by an observant reader. For the sake of this demonstration, it will be assumed that a visual scan was unable to isolate the offending data.

Even a neophyte programmer can easily see that 2 of the 3 lines being combined contain freeform text, while the first line contains freeform text as well as two numeric fields. The first exploratory ad hoc will take advantage of that fact. The ad hoc will keep an internal line counter loop, running from 1 to 3, then resetting the counter to 1 and repeating the process. It will assume that the 1st of each 3 line group should contain numeric fields corresponding to Birth and Death dates, and will validate that using the VERIFY function. (Yes, there are other ways to handle this validation; the reader can select their favorite.) Any non-numeric values found where these two fields are expected will be captured and written to the SASLOG.

```
149  DATA  _NULL_;
150     RETAIN  LINENUM 1;
151     INFILE  'NEXTLINE DATA2 A'  END=LASTREC;
152     INPUT   @ 41  BORN_C   $CHAR4.
153             @ 46  DIED_C   $CHAR4. ;
154     BORN_V = VERIFY( BORN_C, '0123456789' );
155     DIED_V = VERIFY( DIED_C, '0123456789' );
156     IF  LINENUM = 1  THEN  DO;
157        IF  BORN_V  OR  DIED_V  THEN  DO;
158           FILE  LOG;
159           PUT 'Expected Birth/Death not numeric '
160                _N_=  BORN_C=  DIED_C= ;
161        END;
162     END;
163     IF  LINENUM = 3  THEN  LINENUM = 1; /*reset count*/
164     ELSE  LINENUM = LINENUM + 1;    /*increment count*/
165  RUN;
NOTE: The infile 'NEXTLINE DATA2 A' is:
      File Name=NEXTLINE DATA2 A,
      Lrecl=80,Recfm=FB,Blksize=960
Expected Birth/Death not numeric _N_=10 BORN_C=1943 DIED_C=
Expected Birth/Death not numeric _N_=13 BORN_C=1966 DIED_C=
NOTE: 17 records were read from the infile
     'NEXTLINE DATA2 A'.
```

The ad hoc displays two offending records. Unfortunately, upon further review, this first attempt only demonstrates an error in the original assumption. The data does not contain a year of death for those individuals who are still living, leaving the value blank. The ad hoc does not account for this situation. The first rash solution might be to add a blank to the list of valid values that the routine can accept in this field. However, that solution would resolve the problem at hand, while introducing more issues. For example, it would allow two-digit years, which may not be valid for this particular data. Worse, it would assume embedded blanks, such as “1 45”, to be acceptable. Further thought allows a more comprehensive solution - the ad hoc will be modified to treat blank values for this field as valid.

```
177  DATA  _NULL_;
178     RETAIN  LINENUM 1;
179     INFILE  'NEXTLINE DATA2 A'  END=LASTREC;
180     INPUT   @ 41  BORN_C   $CHAR4.
181             @ 46  DIED_C   $CHAR4. ;
182     BORN_V = VERIFY( BORN_C, '0123456789' );
183     DIED_V = VERIFY( DIED_C, '0123456789' );
184     IF  LINENUM = 1  THEN  DO;
185       IF  BORN_V OR (DIED_V AND DIED_C¬=" ") THEN  DO;
186         FILE  LOG;
187         PUT  'Expected Birth/Death not numeric '
188              _N_=  BORN_C=  DIED_C= ;
189       END;
190     END;
191     IF  LINENUM = 3  THEN  LINENUM = 1;
192     ELSE  LINENUM = LINENUM + 1;
193  RUN;
NOTE: The infile 'NEXTLINE DATA2 A' is:
      File Name=NEXTLINE DATA2 A,
      Lrecl=80,Recfm=FB,Blksize=960
NOTE: 17 records were read from the infile
      'NEXTLINE DATA2 A'.
```

This second attempt successfully eliminates the false positives from our test. Unfortunately, the SASLOG now displays NO helpful messages! It might be assumed that all of the 1st records are present and in the correct order. Of course, it is also possible that some of the free-form text also contain valid 4-digit numeric values in the same columns that the first record stores the two dates. The analyst must make a decision at this point. He or she can continue on the current path and enhance the ad hoc to verify that the other records do not, in fact, coincidentally contain valid numeric values, or they can put this assumption aside and attempt another solution.

(As an aside, it should be noted that this ad hoc is not complex enough to handle anything other than the first occurrence of a date-related error. The ad hoc never wavers from the assumption that each group contains 3 and only 3 records, even after that theory is invalidated. To be truly useful, this routine should be enhanced to check all records for occurrence of valid dates in the proper columns, to determine if the values are coincidental or if an error has been found, and to adjust the 3-record loop once an error is found and reported.)

The next data condition that can be verified is that each of the 3 lines has some sort of unique identifier. In this contrived example, the records are clearly identified using numbers from 1 to 3. The analyst will be lucky to encounter something this basic in the real world – at least when dealing with problematic data. Often, it will take some combination of fields / values to uniquely identify each line.

The next ad hoc will build upon the assumption that each line is sequentially numbered in a 1 through 3 looping scheme. It will start with the belief that the first line will be numbered ‘1’, and each subsequent line will follow the looping pattern, resetting itself back to “1” again after the 3rd line. Should a breach in that pattern be detected, the routine will display the offending record in the SASLOG, and will reset its internal counter based on the value detected. (This ad hoc is also coded in a manner that is too trusting of its obviously erroneous input data, since it assumes that all fields will contain only the value ‘1’, ‘2’, or ‘3’ – no other numeric values, and no letters or special characters. The ad hoc author must make careful decisions as to how complex to make each routine, remembering that the goal of the ad hoc is simply to provide information to the analyst about their input data.)

```
235         DATA  _NULL_;
236            RETAIN  EXPECT_LINENUM  1;
237            INFILE  'NEXTLINE DATA2 A';
238            INPUT   @  6  LINENUM           1. ;
239            IF  LINENUM ¬= EXPECT_LINENUM  THEN DO;
240               FILE  LOG;
241               PUT  'BADDATA: INPUT LINE=' _N_
242                    'Actual Line Number='  LINENUM
243                    'Expected='   EXPECT_LINENUM ;
244            END;
245            IF  LINENUM = 3  THEN  EXPECT_LINENUM = 1;
246            ELSE  EXPECT_LINENUM = EXPECT_LINENUM + 1;
247         RUN;
NOTE: The infile 'NEXTLINE DATA2 A' is:
      File Name=NEXTLINE DATA2 A,
      Lrecl=80,Recfm=FB,Blksize=960
BADDATA: INPUT LINE=17 Actual Line Number=3 Expected=2
NOTE: 17 records were read from the infile
     'NEXTLINE DATA2 A'.
```

The results of this ad hoc are much more successful than the earlier attempt. Looking at the SASLOG, it can be determined that only one line of the input data is incorrect – the 17th line of the file. Further, it can be seen that this line contains a Line Number of 3, whereas the looping pattern would indicate that the data *should* contain a Record Type of 2. Further, since the SASLOG reports that problem occurs in Line number 17, and there are only 17 lines in the file, it can be determined that the last combination of records is missing the middle record of the triad.

In this particular example, a little extra examination of the original SASLOG could have potentially saved us a lot of work. It could easily be assumed that the last group of 3 records is the one – the only one – with issues. This is because, as established earlier, the variables BORN and DIED are numeric values. However, the SASLOG never indicates that there is an issue with any of the values used to populate those fields. If there were problems with missing or extra records in the file, it is highly likely that the SAS routine would attempt to load non-numeric values into these variables. This would cause additional problems and messages that would have made a much less attractive SASLOG, but one much easier to debug. This knowledge would have saved a great deal of time in researching the situation (but it would have made for a rather short presentation!) Again the goal is NOT to write the prettiest, most effective ad hoc, but to spend as little time as possible in determining the problem and therefore looking towards resolution.

The last part of the puzzle – the solution – is not as straightforward as one might initially assume. There are several potential solutions to the puzzle; any one or several may be appropriate:

- The data sent were incorrect; the provider must recreate and resend.
- The data sent were incorrect. However, corrected data cannot be delivered in a timely manner. The original routine must be enhanced to watch for offending data conditions, either rejecting the offending records outright or using some intelligent assumptions to obtain as much valid data as possible from those records.
- The data sent are correct, but the specifications delivered to the programmer were incomplete – or the routine was not coded correctly. The original routine must be debugged or modified to adhere to the correct conditions.
- The data are and will continue to be of inconsistent quality. The original routine must be enhanced to assist in catching and reporting problematic data without having to take extra time to evaluate the situation after execution.

CONCLUSION

This paper addressed a message that is commonly found in a SASLOG. It discussed the use of ad hoc routines to explore WHY the message 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 one's data and make the readers' daily job easier are the *true* lessons that the reader retains from this paper.

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. (2003), "The BEST. Message in the SASLOG". *Proceedings of the Twenty-Eighth Annual SAS Users Group International Conference*. Cary, NC: SAS Institute, Inc.

Kuligowski, Andrew T. (2000), "Pruning the SASLOG – Digging into the Roots of NOTEs, WARNINGs, and ERRORs". *Proceedings of the Twenty-Fifth Annual SAS Users Group International Conference*. Cary, NC: SAS Institute, Inc.

SAS Institute, Inc. (2000), *SAS OnlineDoc, Version 8.* Cary, NC: SAS Institute, Inc.

The author can be contacted via e-mail at:

KuligowskiConference@gmail.com (preferred) or
KuligowskiAndrew@gmail.com

ACKNOWLEDGMENTS

The author wishes to offer his thanks to Debbie Buck for her assistance in getting this paper from concept to completion. He would also like to give a nod to Justin Hayward, John Lodge, Graeme Edge, Ray Thomas and Mike Pinder – the Moody Blues – whose 1968 album inspired the title of this presentation. (Furthermore, he rather deviously wonders how many music fans will accidentally stumble upon the web site containing this presentation as a result of a web search on the band's or artists' names?)