Testing a computer program is a task that sometimes takes as long as designing the program and writing the code; sometimes it takes twice as long; and sometimes it comes back to haunt us long after the program has been released for production. In the past ten years the testing of computer programs has been simplified. First, computers have become so fast that programmers have the luxury of debugging their work almost immediately after it has executed. In a few hours, numerous combinations of data values can be tested. Second, programming language syntax has evolved into an English-like sentence structure, making it easier to write code and theoretically to debug it. Yet, with both of these dramatic improvements in the computer programmer's environment, the style or methods for thorough testing of programs has not changed. When a programmer is faced with a perplexing error, he may still try a number of futile approaches to resolve the problem with no true understanding of its cause. While we have come a long way in the last 25 years from saying, "It compiled, it's ready", we still have not developed methodical strategies for testing code and for resolving both the obvious and the obscure errors.

In recent years, a set of convenient debugging tools has been developed for many languages. Code tracers and debugging dumps or core snapshots facilitate programmers' attempts to resolve persistent "bugs". However, these tools are only available in a "testing" environment.

Most programming languages do not provide any automatic accounting of the amount of data read, nor do they describe the physical attributes of input and output files. Methodical testing becomes a process of turning "on" and "off" the program dumps. The SAS system, however, is different because the SAS LOG always contains an adequate audit of major data transactions and transformations.

This tutorial will address the interpretation of some of the messages on the SAS LOG. The problems associated with methodical testing of SAS-based programs are lessened because of the kinds of information which are always present in the LOG. A better understanding of these messages — ERRORS and NOTES — will enable SAS programmers to debug their code more expeditiously and to develop efficient strategies for testing their programs. The topics to be addressed in the sections that follow are:

- A systematic approach to reading the SAS LOG.
- Messages associated with reading and writing data.
- Data problems which generate NOTES or ERRORS.
- Detecting logical errors — strategies for successful testing.

A SYSTEMATIC APPROACH TO READING THE SAS LOG

The LOG is the key to how well the SAS Supervisor performed what the program told it to do! The simple informative messages at the beginning of the SAS LOG and the NOTES after each DATA or procedure step can provide important clues to the existence may be occurring, as well as their causes and solutions of problems.

Figure 1 presents standard SAS LOG output for a job which executed with no errors. The first NOTES to appear on the LOG identify the version of SAS under which the job was run, and the location and site number of the computer installation. These entries are important: the version defines the release of the SAS System that the job was run under; the site number is required by the SAS Institute when servicing technical problems and answering questions.

At the completion of each DATA and PROC step, additional NOTES are printed in the LOG. There is a line of information that corresponds to each SAS data set created in the step which details the number of observations, the number of variables per observation, and the number of observations which can fit on one track of disk storage. There is also a NOTE which tells how much time the DATA or PROC step took, to compile and execute and how much memory was required. If a PROC step produced printed output, the pages...
where the report is located are also listed. These page numbers correspond to the numbers on the upper right-hand corner of pages printed after the SAS LOG.

It is important to check these few simple notes carefully. Were all of the data records processed? Were all of the variables read from each record? Can the number of variables output to the newly-created SAS data set and the number records written to or read from SAS data sets or external files be accounted for?

If the step was not executed because of compilation errors (i.e. syntax), there will be several "CLUES":

```sas
DATA NULL;
A = 20;
B = 40; /\* COMMENT IMPROPERLY ENDED /*
IF A < B THEN
DO: /\* ENDS COMMENT, 'IF'...THEN' AND 'DO' IGNORED */
Y = A;
Z = B;
END: /\* SAS FINDS NO MATCHING 'DO' */
ERROR:
324

324: NO MATCHING DO STATEMENT.
```

NOTE: SAS STOPPED PROCESSING THIS STEP BECAUSE OF ERRORS.
NOTE: SAS SET OPTION OBS=0 AND WILL CONTINUE TO CHECK STATEMENTS.
THIS MAY CAUSE NOTE: NO OBSERVATIONS IN DATA SET.
In this case, the SAS Supervisor could not convert the SAS program text into machine-readable code, and so it could not execute the step. The number of variables is correct for all correctly referenced SAS data sets, but the number of observations processed is set to zero. If a PROC step was not executed, there will be no reference to printed output in the NOTE.

Whenever the SAS Supervisor cannot compile a step because of syntactical errors, additional LOG NOTES are printed. The errors are underlined in the program text and identified with a number that defines the error. A full explanation of the error is printed with the NOTES pertaining to that step. Figure 2 presents some of the more common errors. A full listing of all of the numbered errors printed by the SAS System can be obtained by executing "PROC SASHELP;". One error can cause another code to be falsely interpreted which, in turn, may give rise to other errors. A missing semi-colon (;) after the DATA statement causes the INFILE statement and associated DDname to be read as SAS Data Set names. The SAS Supervisor then indicates that there is no INFILE statement.

At the end of the SAS LOG, the following message appears:

NOTE: SAS INSTITUTE INC.
SAS CIRCLE
PO BOX 8000
CARY, N.C. 27511-8000

FIGURE 3

This message does not mean that the job executed successfully, and it does not mean that all the DATA and PROC steps in the job compiled and executed without errors. It only means that the job ended with a normal system completion code - it did not ABEND.

If errors were noted in the LOG, the pages (enumerated on the upper left-hand corner of the SAS LOG) where errors occurred are listed just before the concluding message. All ERROR messages should be accounted for, as a program may have several independent sources of error. Figure 4 presents a SAS job which ran to completion, with errors in the compilation of the program text.

MESSAGES ASSOCIATED WITH READING AND WRITING DATA

SAS generates two types of messages in conjunction with the reading or writing of data in INPUT, PUT, SET or MERGE statements.

First, there are the routine NOTES generated when SAS data sets are created or when data is written to an external file. The messages associated with SAS data set creation have already been described. When the SAS Supervisor writes to an external file (with a FILE statement followed by a PUT statement), a NOTE is generated telling how many lines were written to the file. If the file was on a storage device such as a disk or a tape, this note will be preceded by another NOTE which gives the name, device identification, record length, blocksize, and format of the file being read from. Similar notes are generated when data are read from an external data set.
NOTE: THE JOB DISATEM HAS BEEN RUN UNDER RELEASE 28.3 OF SAS AT OPTIMUM SYSTEMS INC. (02155001).

NOTE: CPUID VERSION = 23 SERIAL = 020498 MODEL = 3081.

NOTE: CPUID VERSION = 23 SERIAL = 020498 MODEL = 3081.

NOTE: SAS OPTIONS SPECIFIED ARE:
NOSVCO NOVATE,S=72,NOOVP,NOCENTER,LS=70 SORT=4

DATA WORKERS;
INFILE WORKREC;
INPUT SEX $ 6 YRS_ED 8-9 EMPLOYED $ 11 PROFESSION $ 13;
ERROR: 103
A NAME CANNOT HAVE MORE THAN 8 CHARACTERS.

NOTE: SAS STopped PROCESSING THIS STEP BECAUSE OF ERRORS.
NOTE: SAS SET OPTION OB5=0 AND WILL CONTINUE TO CHECK STATEMENTS.
NOTE: 0 LINES WERE READ FROM INFILE WORKREC.
NOTE: DATA SET WORK. WORKERS HAS 0 OBSERVATIONS AND 4 VARIABLES. 3128 0.
NOTE: THE DATA STATEMENT USED 0.06 SECONDS AND 284K.

RUN;
PROC FREQ DATA=WORKERS;
TABIES SEX * YRS_ED * EMPLOYED /LIST;
ERROR: ERRORS ON PAGES 1.
NOTE: SAS INSTITUTE INC.
SAS CIRCLE
PO BOX 8000
CARY, N.C. 27511-8000

FIGURE 4
NOTE: FILE OUTFILE IS:
UNIT=DSAD.WO VOL=SER=TEPP01.DISPFH=R.
DCB=(DLKSIZE=3120.RECL=60.RECFM=F9)
NOTE: 10 LINES WERE WRITTEN TO FILE OUTFILE.

FIGURE 5

The second type of messages are generated if certain anomalies are encountered when data is read or written.

The first example shows what happened when SAS did not find the anticipated data on one record. List input was used and on two of the input records, one of the variables on the INPUT statement was missing and not replaced with a "*". To satisfy the variable list on the INPUT statement a second record was read before an observation was output to the SAS data set. Observe the NOTE generated and the NOTES which show that six lines were read from an external file, but only four observations were output to the SAS data set. In cases where the problem is not obvious from the output, these notes may be the only indication that a problem exists. The problem can also occur when the SAS Supervisor reads past the end of a record. In either case, the problem can be avoided by use of the MISSOVER option on the INFILE statement. When the MISSOVER option is in effect only one record is read, and the values of all variables not accounted for are set to missing.
When SAS data sets are merged with BY variables, a NOTE will be generated when, for a level of the BY variable(s), there are multiple records in more than one of the SAS data sets being merged. Since the MERGE is often used for table lookup, the programmer usually intends that only one of the data sets in the MERGE statement should have multiple records level of BY variables.

When the SAS program uses a PUT statement with a format, a NOTE will be generated if the format for a numeric variable is too small to accommodate the variable. The Supervisor will try to find an alternate way to fit the value into the space with minimum loss of precision. The NOTE shown below warns the programmer that some of the output will not be in the format anticipated, a problem situation when an external file is written to a storage device in machine-readable form.

### Table: MARKUP RATES

<table>
<thead>
<tr>
<th>OBS</th>
<th>ITEM</th>
<th>MARKUP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BOOK</td>
<td>0.10</td>
</tr>
<tr>
<td>2</td>
<td>DRESS</td>
<td>0.45</td>
</tr>
<tr>
<td>3</td>
<td>DRESS</td>
<td>0.65</td>
</tr>
<tr>
<td>4</td>
<td>HIFI</td>
<td>0.40</td>
</tr>
<tr>
<td>5</td>
<td>HIFI</td>
<td>0.40</td>
</tr>
<tr>
<td>6</td>
<td>TOASTER</td>
<td>0.25</td>
</tr>
</tbody>
</table>

### Table: PURCHASES

<table>
<thead>
<tr>
<th>OBS</th>
<th>ITEM</th>
<th>PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BOOK</td>
<td>15.95</td>
</tr>
<tr>
<td>2</td>
<td>DRESS</td>
<td>40.00</td>
</tr>
<tr>
<td>3</td>
<td>DRESS</td>
<td>60.00</td>
</tr>
<tr>
<td>4</td>
<td>DRESS</td>
<td>120.00</td>
</tr>
<tr>
<td>5</td>
<td>HIFI</td>
<td>600.00</td>
</tr>
</tbody>
</table>

```sas
DATA IND PROF(KEEP=ITEM PROFIT MARKUP);
  MERGE TABLE PURCHASE(IN=BOUGHT);
  BY ITEM;
  IF BOUGHT THEN
    DO;
      PROFIT = PRICE*MARKUP;
      OUTPUT IND_PROF;
    END;
```

### Figure 6

**NOTE:** The data set used 0.07 seconds and 234K.**

### Figure 7

**NOTE:** The data set used 0.07 seconds and 234K.

### Figure 8

**NOTE:** At least one W.D format was too small for the number to be printed. The decimal point may be shifted by the "BEST" format.
When SAS reads data from an external file with an INPUT statement and a field proves to contain non-numeric characters, the variables read from that field will have values set to missing. Similarly, when numeric data are required, and a variable which was previously defined as character is used in the expression, a character to numeric conversion takes place. In this case, however, character strings which contain non-numeric characters are evaluated as "missing" values. The LOG message appears as above. Following the message, the SAS System details where the error occurs in the program text and how many times it occurs (line number: column number). An examination of the program text should determine whether the variable was incorrectly defined, or the expression was in error.

Whenever a variable's value is "missing", the use of that variable in a mathematical operation causes the result of the operation to be set to "missing".

When the expression is written as \( z = \text{SUM}(A, B) \), all variable values that were "missing" are ignored. This expression is referred to as the SUM function. The mathematical functions such as \( \text{SUM} \), \( \text{MEAN} \), \( \text{STD} \), also ignore missing values.

NOTE: NUMERIC VALUES HAVE BEEN CONVERTED TO CHARACTER VALUES AT THE PLACES GIVEN BY: (LINE):(COLUMN).

FIGURE 10
No programming language can perform illegal mathematical computations. These include division by zero, taking the logarithm of zero and taking the square root of a negative number. The SAS System deals with such occurrences by setting the result of the expression to a "missing" value.

Again, the NOTE is very explicit as to where the computation took place and how often the event occurred.

During the execution of some procedures, the memory limit may be reached. This can happen, for example, in procedures like FREQ, SUMMARY and TABULATE when there are too many values for classification variables (i.e. the data points are continuous rather than discrete). The following message appears:

```
ERROR: NOT ENOUGH MEMORY FOR ALL VARIABLES. VARIABLE NUMBER REMOVED AFTER 2779 LEVELS.
```

It is important to understand that the presence of any of these NOTEs does not terminate the execution of the SAS job. The SAS System takes its own corrective action, prepares the NOTE for the LOG, and continues to process the DATA or PROC step. This does not mean that the results of the job are valid nor that they are invalid. Computational errors must, however, be accounted for and reconciled.

Data problems can also occur during the execution of procedure. Whenever BY-group processing is specified, and the data are not sorted on the BY variables, the following messages appear:

```
ERROR: THE DATA SET WORK.HMLIST IS NOT SORTED BY VARIABLES IN BY LIST.
TO SORT THE DATA SET USE PROC SORT.
IF YOUR DATA IS IN GROUPS USE KEYWORD NOTSORTED IN BY LIST.
ERROR: STATUS SET.
NOTE: ABOVE MESSAGE FOR BY GROUP:
NAME=ELLEN
```

Some printed output from the procedure may be present because execution of the PROC step does not stop until an observation is found to be out of order. If this happens after several values of the BY variables have been processed, some output will be printed. All of the message associated with each PROC step must be read carefully because the presence of output is no guarantee that all of the data were processed.

Detecting Logical Errors
Strategies for Successful Testing

Logical errors are by far the most difficult to detect. They may surface long after the program has been put into use. In many cases, no ERRORs and no extraordinary NOTEs will be generated, and the printed output may appear normal.

```
ERROR: NOT ENOUGH MEMORY FOR ALL VARIABLES. VARIABLE NUMBER REMOVED AFTER 2779 LEVELS.
```

One logical error which gives rise to an ERROR message is the case in which the variable used as index for an array has a value beyond the bounds of the array (i.e. less than one or more that the number of elements in the array) at the time the array is referenced.

```
ERROR: THE DATA SET WORK.HMLIST IS NOT SORTED BY VARIABLES IN BY LIST.
TO SORT THE DATA SET USE PROC SORT.
IF YOUR DATA IS IN GROUPS USE KEYWORD NOTSORTED IN BY LIST.
ERROR: STATUS SET.
NOTE: ABOVE MESSAGE FOR BY GROUP:
NAME=ELLEN
```

```
ERROR: THE DATA SET WORK.HMLIST IS NOT SORTED BY VARIABLES IN BY LIST.
TO SORT THE DATA SET USE PROC SORT.
IF YOUR DATA IS IN GROUPS USE KEYWORD NOTSORTED IN BY LIST.
```
When this error occurs, no more observations will be processed, and it must be corrected in order for the program to run to completion. The information in the message should help the programmer to determine whether the error was caused by bad data or by incorrect program logic.

The following case gives little indication that a problem exists in the logic of the program. A subsetting IF or a DELETE statement preceding a statement which checks a FIRST, LAST, or last observation condition may cause unexpected results. Any processing which should occur when these conditions arise will not take place. If such processing includes outputting to a SAS data set, the data set may have fewer observations than expected. Likewise, lines conditionally printed to the SAS LOG or to external files with PUT statements may not be printed. In the example below, the final observation was deleted, and so the line which should have been printed in the SAS LOG was omitted.

The SAS LOG can be even more useful if diagnostic PUT statements are used to list data where potential problem condition exist. A PUT _INFILE_ statement will print the input buffer (a mirror image of the entire last record read from an external file), not merely those fields read in the INPUT statement. A PUT _ALL_ statement will print all variables in the program data vector with their current values. Unless specified with a FILE statement, the output of these statements will be printed in the SAS LOG at the end of the DATA step.

In addition, the use of PROC FREQ or PROC TABULATE is recommended before and after a DATA step in which major data transformation takes place, to determine whether the program performed the transformation correctly.

```
DATA _NULL_;
ARRAY _VAR (1) VAR1 - VAR9;
DO I = 1 TO 10;
   TOTAL + _VAR;
END;
ERROR: ARRAY SUBSCRIPT OUT OF RANGE AT LINE 24 COLUMN 15.
I=10 VAR1=, VAR2=, VAR3=, VAR4=, VAR5=, VAR6=, VAR7=, VAR8=, VAR9=.
TOTAL=9, ERROR=. I=10
NOTE: SAS SET OPTION ODS=O AND WILL CONTINUE TO CHECK STATEMENTS.
THIS MAY CAUSE NOTE: NO OBSERVATIONS IN DATA SET.
FIGURE 13
```

```sas
When this error occurs, no more observations will be processed, and it must be corrected in order for the program to run to completion. The information in the message should help the programmer to determine whether the error was caused by bad data or by incorrect program logic.

The following case gives little indication that a problem exists in the logic of the program. A subsetting IF or a DELETE statement preceding a statement which checks a FIRST, LAST, or last observation condition may cause unexpected results. Any processing which should occur when these conditions arise will not take place. If such processing includes outputting to a SAS data set, the data set may have fewer observations than expected. Likewise, lines conditionally printed to the SAS LOG or to external files with PUT statements may not be printed. In the example below, the final observation was deleted, and so the line which should have been printed in the SAS LOG was omitted.

The SAS LOG can be even more useful if diagnostic PUT statements are used to list data where potential problem condition exist. A PUT _INFILE_ statement will print the input buffer (a mirror image of the entire last record read from an external file), not merely those fields read in the INPUT statement. A PUT _ALL_ statement will print all variables in the program data vector with their current values. Unless specified with a FILE statement, the output of these statements will be printed in the SAS LOG at the end of the DATA step.

In addition, the use of PROC FREQ or PROC TABULATE is recommended before and after a DATA step in which major data transformation takes place, to determine whether the program performed the transformation correctly.
```

```
DATA SET ONE
OBS STUDENT SUBJECT
  1 JONES HISTORY
  2 JONES PHYSICS
  3 JONES ART
  4 KING SPANISH
  5 PETERS GERMAN
  6 PETERS SPEECH
  7 PETERS ALGEBRA
  8 PETERS BIDLOGY
  9 SMITH CALCULUS

DATA _NULL_;
SET ONE END=LASTREC;
IF SUBJECT = 'ALGEBRA'
OR SUBJECT = 'CALCULUS'
THEN DELETE;
PUT STUDENT SUBJECT;
NOTMATH = 1;
IF LASTREC THEN
PUT 2 5 'NON-MATH ENROLLMENT = NOTMATH .
JONES HISTORY
JONES PHYSICS
JONES ART
KING SPANISH
PETERS GERMAN
PETERS SPEECH
PETERS BIOLOGY
NOTE: THE DATA STATEMENT USED 0.95 SECONDS AND 234K.
FIGURE 14
```
CONCLUSION

This paper has presented users of the SAS system with a methodical approach to reading and understanding the SAS LOG. Success in programming is measured by the thoroughness, accuracy, and reliability of the programs that one writes. Careful debugging and testing greatly enhance one's skills as a programmer. A programming error that is accurately corrected in a systematic manner will provide a well-learned lesson.

The major points of this paper can be summarized as follows:

- Read the SAS LOG very carefully. The messages in the LOG provide a guide for verifying the results of the program that was executed.

- The routine NOTES at the conclusion of each step describe how the SAS Supervisor read the program text and how it processed the data.

- All ERRORS are clearly detailed with an explanation. Some ERROR messages, however, may result from preceding errors, which cause a misinterpretation of the text, such as a missing semicolon.

- Be sure that the corrections made to the program text address all the problems raised in the NOTES and ERROR messages.

- ERRORS encountered when data are read do not always result in the abnormal termination of the SAS program. Messages associated with I/O processing may be very explicit when a dump of the input buffer and program data vector are printed. At other times information in the message can be elusive as in the case of the MISSOVER issue or when BY variable values result in an incorrect MERGE.

- Computational errors and computations on missing values can be handled with extra programming to detect and prevent anticipated errors, so that only "unexpected" errors will appear.

- Detection of logical errors is an area in which the SAS Supervisor does not give obvious help. However, by using debugging tools such as PUT _INFILE_ or PUT _ALL_, the programmer can create a trace of the data step execution.

Reading the LOG, bearing in mind all of the details mentioned above, and resolving errors so that all of the LOG messages can be accurately accounted for, will strengthen one's SAS programming skills and provide the discipline necessary to solve many types of programming problems.

Authors can be contacted at:

ORI, Inc.
122 C Street, NW
Suite 250
Washington, DC 20001
(202)737-2666

*SAS is a registered trademark of SAS Institute, Cary, NC, USA.