GETTING THE MOST FROM THE SAS® LOG

Robert G. Hall and Judith H. Mopsik
Computer Sciences Corporation
Rockville, Maryland

The LOG is the most useful debugging tool available to the SAS programmer. The LOG indicates how the SAS Supervisor compiled and executed the code that was submitted. The LOG, as appropriate for an interpretative language such as SAS, combines the normal syntax information that might be obtained from a compiler with additional information about what files are created and which major routines (PROCS) are executed.

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 short explanation of the error is printed with the NOTEs pertaining to that step. A full listing of all of the numbered errors printed by the SAS System can be obtained by executing "PROC SASMSG;". One error can cause further code to be falsely interpreted which, in turn, may give rise to other errors. For example, 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!

This paper describes ways to use the LOG in your programming.

POOR USE OF THE LOG

As with any information, the LOG is only valuable if you use it. At one extreme, you don't have to look at the LOG at all. You can simply examine your output file and guess whether the program executed correctly or not. If your program is not complex and you are familiar with the data, there is a finite and probably high probability that you will be able to detect whether the program worked or not.

If you are faced with a more complex problem, you can still avoid an indepth examination of the LOG. You submit a SAS batch job or locally submit an interactive job; it executes; and, using a system provided text editor, you search the LOG for the presence of the character string "ERROR" or in an interactive environment you would look for red error lines scrolling past. If you don't find an instance of "ERROR", then the job ran correctly. Now you can safely say that you looked at the program output—it looks correct—and the program LOG—found no errors—therefore, the program is correct.

Unfortunately this may not be an uncommon occurrence. For example, you develop a large SAS application as a single program and have to test it; or you develop a complex system of macros which call each other in a clever scheme which is very difficult to trace; or you inherit a large program and have to make a number of changes to it. Many programmers don't have the fortitude to sit down and work their way through a 1000 lines of SAS LOG verifying each step of the way. They look for obvious errors; if there are none and the printed report looks correct, then the job is finished.
This is poor programming technique no matter how structured your code is, no matter how many comments you included, no matter how much indentation and white space you included. You must look at the LOG in detail if you want to be sure that your code is correct.

**DEBUGGING USING THE LOG**

**Errors**
SAS lists all ERRORs with a detailed explanation. Solving the first error is usually not difficult. Some ERROR messages, however, may result from preceding errors, which may cause a misinterpretation of following code, such as a missing semi-colon. Be sure that the corrections made to the program text address all the problems raised in the NOTEs and ERROR messages. Unfortunately, early mistakes may completely mask later code—causing an error correction cycle to repeat. The correction of one error early in the program, will often reveal new error messages in later programming steps.

**Tracing (SAS provided information)**
During debugging, the LOG, of course, contains all the information SAS provides you about the program you submitted. SAS generates two types of messages in conjunction with the reading or writing of data in INPUT, PUT, SET, or MERGE statements. There are routine data set related NOTEs and there are computation NOTEs. NOTEs are also included containing information about what PROCs were executed.

There are the routine NOTEs generated when SAS data sets are created or when data is written to an external file. The data set NOTEs detail exactly what data sets were created, how many variables they had, and how many observations each data set had. 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.

Data set NOTEs are very useful in tracing the data flow in the program. The programmer should check at a minimum:
- The correct data sets are created at each step
- These data sets have the correct variables. To enhance your ability to trace data flow, you should use good programming practices and be very explicit concerning the variables that you KEEP in your data sets. Tracing ten variables is much easier than tracing 2056 variables.
- The data sets have the right number of observations: Is your merge working correctly? Is your subsetting working? In order to validate your program, you should independently determine what the number of observations your program should be including at each step of the process. This, in turn, means that you must have an in-depth knowledge of your test data.

Computations which cannot be executed as written generate NOTEs in the LOG. These computations fall into three main categories:
- numeric to character, or character to numeric conversions
- mathematical operations for variable values that are "missing"
- illegal mathematical operations.

SAS is very kind to you; it will convert apples into oranges and add them together; but the conversion may not be to your liking. For example, SAS will use the BEST12.0 format to convert character to numeric and that may not be appropriate. For character to numeric conversion, use the INPUT function. Use the PUT function for numeric to character conversion. For example:

```plaintext
char = put(numeric,2.0);
numeric = input(char,2.0);
```
You should use good programming practices to explicitly handle all computational errors and computations on missing values—to detect and prevent anticipated messages—so that only "unexpected" problems appear in the LOG.

Rather than:

```
x = y*z;
```

Use:

```
if y =. or z =. then x =.;
else x = y*z;
```

Good programming practices will also include code to ensure that you don’t try to execute code which cannot be executed.

Rather than:

```
x = y/z;
```

Use:

```
if z = 0 then x =.;
else x = y/z;
```

You can use the LOG to follow good programming practices by ensuring that these "informational" messages don’t appear.

Tracing (User provided information). The LOG can also contain information that the programmer inserts into the LOG to help in debugging.

- Sometimes, it is important in debugging to establish that a specific data or program step was executed. This can be very basic:

  data &test;
  PUT "&TEST DATASET created.";
  sas code here...

- Many programmers like to put messages into macros indicating that macro compiler is compiling a specific macro. The use of a TESTING switch will cause these messages only to appear in TESTING and not in production. The following macro can be called whenever you want to send a message to the log.

  ```
  %macro macmsg(msg =);
  %if &testing = yes
  %then %PUT &msg;
  %mend macmsg;
  ```

  It is very important in testing, to ensure that all key program branches are executed. For example,

  ```
  if x = 1 then
  do;
  PUT "Executing first option";
  %macone;
  end;
  else do;
  PUT "Executing second option";
  %mactwo;
  end;
  ```

- There are times when the LOG needs to reflect the values of specific variables. This is accomplished using the PUT statement. This is the same statement used to produce a customized report from a DATA _NULL_ step, but in this case no external file is defined so that it writes directly to the LOG. An explicit:

  ```
  PUT var1 var2 var3;
  ```

  will write out the values of all the named variables. A more readable version is:

  ```
  PUT var1 = var2 = var3 =.;
  ```

  By locating the PUT statement strategically within the data step, you can obtain information about what the values of key variables are before or after the execution of SAS statements.
When the situation is desperate, you sometimes want to simply examine the values of all the Program Data Vector variables. This is accomplished by simply:

```
PUT _ALL_;
```

**Acceptance Testing.**
A full LOG provides a detailed description of the acceptance test that you perform on the code. It documents the test data base, provides listings of all of the code, clearly delineates where the data you are manipulating come from, and together with the appropriate output files proves that the program works.

**THE SAS LOG AND EXECUTION SPEED**
Ordinarily having SAS produce a LOG is good; sometimes it is undesirable--particularly in production systems. Two years ago we were developing an SAS/AF application on an OS/2 platform. The application consisted of a number of complex screens combined with a variety of sometimes time consuming data management submit blocks. After testing that everything was working, we shifted our focus to reducing to a minimum the time of execution.

After creating a RAM disk and moving our SAS work library up to it; after examining the code looking at efficiencies; we were still not happy with the execution time. Although it was significantly faster than the legacy system we were replacing, we still thought it should be faster. The client, who was testing independently on another machine, found something that they wanted changed (program specifications always change even during testing). As we changed it and inserted a PUT statement for testing, we noticed that the writing the test statement to LOG had a surprising impact on the response timing of the system.

We got bold, we set out options to:

- **NOTES** we knew the system worked; the end-user didn’t care what data sets were created
- **NOSOURCE** we knew the code worked; the end-user didn’t need to see it listed
- **NOSOURCE2** we knew the included code worked; the end-user didn’t need to see it
- **NOMPRINT** we knew the macro code worked; the end-user didn’t need to see it

We found that the system was significantly faster. Further the use of the application by the end-user was certainly not impacted. The end-users typically do not care about the LOG; thus any time spent writing to the LOG is wasted CPU time to the end-user. The lesson here is that the LOG was be very useful during development but might actually be detrimental to the use of the application in production.

**NOTES YOU SHOULD WATCH OUT FOR**
The following "NOTEs" mean that you program probably is not working the way you want it to:

- **Invalid data in line...** Your input statement doesn’t match what you expected; who knows what data is getting read into which variable. Correct it before you move further into SAS. This correction may involve reading in a "numeric" value as character, performing processing to account for different encodings of missing values, and converting the remaining data into numeric format. In order to have a validated and accepted program, the programmer should be explicit on how the program should handle exceptions.
SAS went to a new line when input statement. Unless you have multi-line input, then you again have a significant problem in reading in your data. This problem must be corrected through either a corrected INPUT statement or through corrections to your raw data file.

At least one _w.d format was too small. If you don't correct this problem, then either you are not following good programming practices because you are carrying variables you don't need or your program is just plain wrong.

Merge statement has more than one data set with repeats of BY values. If you get this message and you don't change your code, you are asking for trouble. The correct handling of the many-to-many merge is a problem that is very difficult to solve in old-style base SAS code. Fortunately, a relatively simple SQL statement in which you can be explicit on what the merge should look like makes the solution to this problem very simple. If you are ever going to use SQL, use it here.

Mathematical operations would not be performed. This problem probably results from the handling of missing data. You must be explicit about how you want SAS to handle missing data.

PROBLEMS THE LOG DOESN'T TELL YOU ABOUT

There are some problems the LOG doesn't tell you about and this can cause a lot of problems. For example, in some cases it doesn't tell you about operating systems errors that might occur. The most common error we have encountered of this type is in MVS when you have the DISP option incorrect in your JCL. If your DISP is set incorrectly, then MVS will not let you replace your data set--the problem is that the LOG describes the program as having worked. The only clue to this problem is in your JCL printout where a return code of 2 is listed next to the JCL listing describing the data set. The LOG contains no information at all about this problem.

A second problem that the LOG doesn't show logic problems with your application. All the clever/efficient/structured/commented code in the world won't help you if you have logic problems with your code. Logic problems may include incorrect requirement specifications (the data items you have selected to report aren't the items the user wants to see), incorrect assumptions concerning algorithms (the collapsing mechanisms for defining when an adverse events begins and ends may be what you've always used is not what your current client wants to use). If your program and your tests are not closely tied to the program specifications, then the LOG will only tell you that whatever you have written works correctly. It will not say anything about whether it works the way the end-user wanted it to work.

A last major problem that the LOG doesn't show some programming errors--usually errors of omission. I clearly remember the day I discovered that I had neglected to included a BY statement with conjunction with a merge statement. In case you didn't know, the MERGE statement works just fine without a BY statement. Further, you can't distinguish the two from the log output. Certainly, however, the two merge statements have strikingly different resultant output data sets.
CONCLUSION

The LOG is the most useful programming tool available to a SAS programmer. In conjunction with output files it can constitute sufficient information to adequately test your program. Like all other tools, however, it is only useful if you use it. A good programmer will use the LOG to:

- Trace variables and observations to ensure that the data flow is correct
- Ensure that all conversions and missing data are handled correctly and explicitly
- Debug specific trouble spots during development
- Match up program requirements and test data observations into an acceptance test package

Authors can be contacted at:

Computer Sciences Corporation
1301 Piccard Drive
Rockville, MD 20850
301/258-5542 and 301/258-5533