Paper 9341-2016

The Three I's of SAS® Log Messages, IMPORTANT, INTERESTING, and IRRELEVANT

William E Benjamin Jr, Owl Computer Consultancy, LLC, Phoenix AZ.

ABSTRACT

I like to think that SAS® error messages come in three flavors, IMPORTANT, INTERESTING, and IRRELEVANT. SAS calls its messages NOTES, WARNINGS, and ERRORS. I intend to show you that not all NOTES are IRRELEVANT nor are all ERRORS IMPORTANT. This paper will walk through many different scenarios and explain in detail the meaning and impact of messages presented in the SAS log. I will show you how to locate, classify, analyze, and resolve many different SAS message types. And for those brave enough I will go on to teach you how to both generate and suppress messages sent to the SAS log. This paper presents an overview of messages that can often be found in a SAS Log window or output file. The intent of this presentation is to familiarize you with common messages, the meaning of the messages, and how to determine the best way to react to the messages. Notice I said "react", not necessarily correct. Code examples and log output will be presented to aid in the explanations.

INTRODUCTION

This presentation will let you identify important messages and simple comments about your job and SAS environment and aid in understanding the meaning of a message. You will be able to quickly determine your response to SAS messages. Additionally, throughout this presentation we will be discussing several types of SAS messages presented to the programmer/operator that assist in the analysis, correction, and execution of SAS processes. Some messages within the SAS system can be suppressed, some can be generated, and you can make up some of your own.

Again, throughout this presentation I will be discussing messages output by the SAS system. Over the years I have come to view the messages in three ways that I call "I cubed". I use the terms "Important", "Interesting", and "Irrelevant" to describe the SAS messages sent to the output log location. SAS classifies messages into three categories also, but they call them "Error", "Warning", and "NOTE" messages. But be careful, some ERRORS are Irrelevant while some NOTES are Important. Context is everything.

THE PROBLEM

One issue that both beginning SAS users and experienced SAS users have is determining the meaning of SAS messages on the output log. Being unable to determine the meaning of messages forces you spend a lot of time determining what a message means only to find out that the message has no impact or at lease no adverse impact on their job. While often experienced programmers believe that a message is just a note or warning and has no impact on their job. Both of these opinions can often be wrong. We will discuss the following seven topics regarding the SAS Message system. As a SAS Programmer you need to be able to classify, analyze, resolve and locate SAS messages.

- Classifying Messages
- Analyzing Messages
- Resolving Messages
- Locating Messages
- SAS ERROR Message Types
- Suppressing Messages
- Generating Messages

The method we will use to analyze SAS messages today will be to present SAS code, show the output log, and discuss results. Because this is a paper about messages, most images will be of the SAS output log and will include all run time notes generated. We will go through this presentation the same way you would when you are debugging your code, we will look at the code, look at the log, and often try again.

CLASSIFYING MESSAGES

I like to classify the SAS messages sent to the log as one of the three types of messages:

I **3 = Important
Interesting
Irrelevant

SAS labels the messages sent to the log as “Error”, “Warning”, “and “Note”. In the next few pages I will explain myself so you can decide for yourself. While both SAS and I put the messages into three categories, I do not always group the messages the same way as SAS groups them.

LOCATING MESSAGES

Generally speaking the SAS messages are output to the SAS LOG window. When SAS starts the system it outputs messages about the current environment. This generally includes the following:

- SAS Copyright notice
- SAS Version number
- Owner of the License and the site number
- Operating System
- Additional Software information
- Computer resources required to start SAS

When starting SAS using a command line interface the ALTPRINT option will copy all information output to the SAS LOG window into the file identified in the ALTPRINT command for the duration of the job. Additionally, the PROC PRINTTO command will send all further messages to the file identified by the PROC PRINTTO command. Another PROC PRINTTO command will change output location or return output to the SAS LOG.

Most individual messages will be placed directly after the DATA step or PROC that executes while most syntax error messages will appear directly below the SAS line of code that contains the error.

ANALYZING MESSAGES

The object of this paper is to teach you how to locate and interpret the messages that SAS generates when it is processing the log files. This paper will take a three step approach to analyze a message.

- Display SAS code
- Show output Log
- Discuss Results

In other words I will show code that will cause an error then show the log output of the error and discuss how to fix the errors. Some of the examples may need multiple cycles of this process to determine all of the errors.

A NOTE ABOUT CONSERVING SPACE IN THIS DOCUMENT.

I am deleting messages similar to the following at the end of Output Logs:

```
NOTE: There were 395 observations read from the data set SASHELP.SHOES.
NOTE: The data set WORK.SHOES has 395 observations and 0 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

Embedded messages will be replaced with: *…SAS NOTES ABOUT JOB TIMING – SEE ABOVE…*

Output Log – Suppressed Messages to Conserve Space.

EXAMPLE 1 – A NOTE THAT MAY OR MAY NOT NEED ACTION

Let us start with this simple example SAS Code segment:

```
DATA _null_;
   a = 5;
   CALL SYMPUTX('max_array_size',a,'G');
DATA temp;
   array second(&max_array_size) _temporary_ ;
   second(3)=15;
   DO I = 1 to &max_array_size;
    PUT second(I)=;
   END;
PROC PRINT data=temp;
RUN;
```

Output Log 1

```
1    DATA _null_;
2       a = 5;
3       call symputx('max_array_size',a,'G');

...SAS NOTES ABOUT JOB TIMING – SEE ABOVE...
4    DATA temp;
5       array second(&max_array_size) _temporary_ ;
NOTE: The array second has the same name as a SAS-supplied or user-defined
function.  Parentheses following this name are treated as array references
and not function references.
6       second(3)=15;
7       DO I = 1 to &max_array_size;
8        PUT second(I)=;
9       END;

second[1]=.
second[2]=.
second[3]=15
second[4]=.
second[5]=.
...SAS NOTES ABOUT JOB TIMING – SEE ABOVE...
10   PROC PRINT DATA=temp;
11   RUN;
```

Output Log 1 Explanation

I deliberately presented the code above without a text explanation. Most of the notes in this log output follow the code which generated the message after the data step or procedure call is complete. However the note between the code lines “5” and “6” is directly after the SAS code that invoked the message. Because you can name an array anything you want to call it AND SAS has a time related function called “SECOND” (the same name given to the array) This note is telling you that is you use time functions in this data step you may not get the result you expect. The impact of this kind of a message requires a careful examination of your code to determine if you need to take any action to eliminate the message. Note also that in some older versions of SAS the absence of a “RUN” statement between code lines 3 and 4 and between 9 and 10 would have caused the messages to be placed differently in the log.

As I mentioned before SAS classifies messages sent to the log as the following:

- NOTE
- WARNING
- ERROR

Example 1 showed a set of NOTE messages one of which could have been an issue if you were using both an array and a SAS function named “SECOND”. Generally speaking I believe that the following is true SAS log messages.

- Notes are informational and usually REQUIRE NO ACTION, but note messages present useful information that may require action (like pointing to the right file) they are useful and may be pointing out an error in the input files, output files, or program logic.
- Warning messages describe things that may have unintended consequences and are presented to make the programmer aware of possible undesirable conditions that could occur.
- Error Messages are displayed by SAS when certain syntax, environment, control or logical conditions exist that are outside of the expected norms. These messages usually prevent the successful completion of the SAS process in question.

EXAMPLE 2 – NOTES ASSOCIATED WITH FILE ACTIONS

Output Log 2

```
27   LIBNAME aa "c:\my_sas_dir";       /* The following NOTE appears when
an input file does not exist*/
NOTE: Library AA does not exist.
28
29
30   LIBNAME aa "c:\my_sas_code";       /* The following NOTE appears when
an input file does exist   */
NOTE: Libref AA was successfully assigned as follows:
      Engine:        V9
      Physical Name: c:\my_sas_code

31   * Not all file actions produce notes;
32   FILENAME bb 'C:\My_Excel_Files';            /* directory does exist
*/
33
34   FILENAME cc 'C:\My_Excel_Files\shoes.csv';  /* file does exist
*/
35   FILENAME dd 'c:\my_sas_dir';                /* directory does not
exist */
```

Output Log 2 - Explanation for NOTES ASSOCIATED WITH FILE ACTIONS

This output log text shows NOTEs that are generated for LIBNAME and FILENAME statements, when files do and do not exist.

- Line 27 generates a NOTE when a file does not exist.
- Line 30 generates a NOTE when a file does exist.
- Lines 32, 34, and 35 generate no NOTEs regardless of whether or not the file exists.

EXAMPLE 3 – MESSAGES THAT RESULT FROM MERGING DATASETS INCORRECTLY

Given this sample code, we will look at some different ways to merge SAS datasets. Not all of which generate the correct results. This code produces two files “A” and “B”. File “A” has 20 records and file “B” has 16 records. Each file has the variable “i” that ranges from 1 to 10. File “A” contains two records with each value of “i” (1 to 10). File “B” contains two records for “I” = 2, 3, 4, 8, 9, and 10 it also contains four records for the value of “I” = 6. The values of the variables aa, bb, cc, d, e, f, and g are functions of “i” but are not meaningful outside this example:

```
OPTION COMPRESS=binary;
DATA A(KEEP=i aa bb cc) B(KEEP=i d e f g);
DO i = 1 TO 10;
   aa = i*11;    bb = i*12;    cc = i*13;        OUTPUT A;
```

```
    aa = i*19;    bb = i*23;    cc = i*31;          OUTPUT A;
    IF MOD(i,3) = 0 THEN DO;
          d = i+7;  e  = i*2;    f  = i**3; g = i*15; OUTPUT B;
          d = i+13; e = i*29;    f  = i**4; g = i*13; OUTPUT B;
    END;
    IF MOD(i,2) = 0 THEN DO;
          d = i+17;  e = i*42;  f  = i**5; g = i*25; OUTPUT B;
          d = i+53;  e = i*72;  f  = i**6; g = i*18; OUTPUT B;
    END;
 END;
 RUN;
```

Output Log 3

```
1    OPTION COMPRESS=binary;
2    DATA A(KEEP=i aa bb cc) B(keep=i d e f g);
3    DO i = 1 TO 10;
4       aa = i*11;    bb = i*12;    cc = i*13;           OUTPUT A;
5       aa = i*19;    bb = i*23;    cc = i*31;           OUTPUT A;
6       IF MOD(i,3) = 0 THEN DO;
7             d = i+7;   e = i*2;    f  = i**3; g = i*15; OUTPUT B;
8             d = i+13; e = i*29;    f  = i**4; g = i*13; OUTPUT B;
9       END;
10      IF MOD(i,2) = 0 THEN DO;
11            d = i+17;  e = i*42;  f  = i**5; g = i*25; OUTPUT B;
12            d = i+53;  e = i*72;  f  = i**6; g = i*18; OUTPUT B;
13      END;
14   END;
15   RUN;

NOTE: The data set WORK.A has 20 observations and 4 variables.
NOTE: Compressing data set WORK.A increased size by 100.00 percent.
      Compressed is 2 pages; un-compressed would require 1 pages.
NOTE: The data set WORK.B has 16 observations and 5 variables.
NOTE: Compressing data set WORK.B increased size by 100.00 percent.
      Compressed is 2 pages; un-compressed would require 1 pages.
```

Output Log 3 – EXPLANATION MESSAGES GENERATED WHEN CREATING FILES A AND B**.**

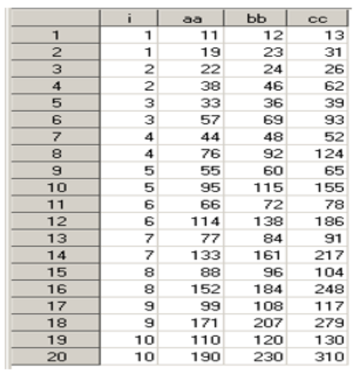
The first two NOTEs are file the file named WORK.A and declare that it has 20 observations, 4 variables, and the binary compression for this file generated a larger file than would have been generated without the effort to compress the file.

The second two NOTEs are file the file named WORK.B and declare that it has 16 observations, 5 variables, and the binary compression for this file generated a larger file than would have been generated without the effort to compress the file.

The last NOTE describes the amount of time the computer used to do the work. The amounts were 0.07 seconds as the clock goes around and the computer Central Processing Unit (CPU) used 0.03 seconds to do the work. This of course indicates that the computer was pre-occupied most of the time doing something else. (0.04 seconds).

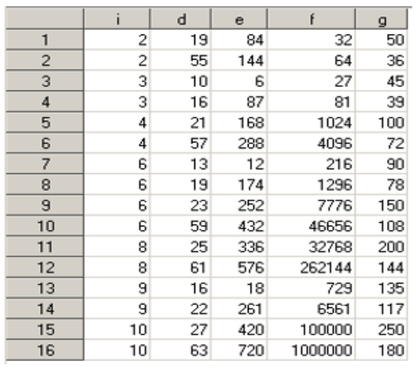
The output files look like this:

File A



	i	aa	bb	cc
1	1	11	12	13
2	1	19	23	31
3	2	22	24	26
4	2	38	46	62
5	3	33	36	39
6	3	57	69	93
7	4	44	48	52
8	4	76	92	124
9	5	55	60	65
10	5	95	115	155
11	6	66	72	78
12	6	114	138	186
13	7	77	84	91
14	7	133	161	217
15	8	88	96	104
16	8	152	184	248
17	9	99	108	117
18	9	171	207	279
19	10	110	120	130
20	10	190	230	310

File B

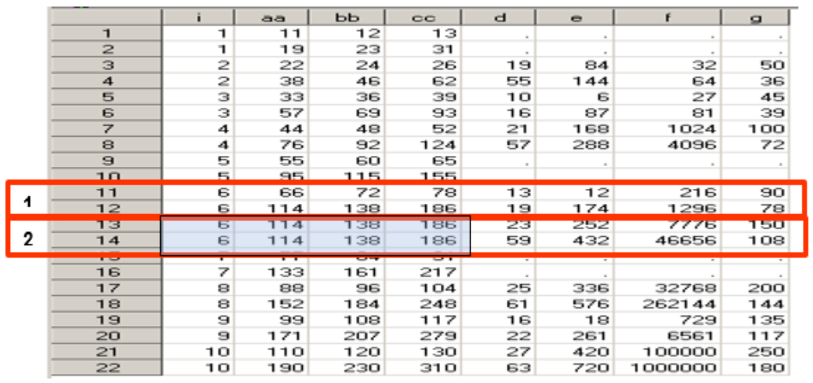


	i	d	e	f	g
1	2	19	84	32	50
2	2	55	144	64	36
3	3	10	6	27	45
4	3	16	87	81	39
5	4	21	168	1024	100
6	4	57	288	4096	72
7	6	13	12	216	90
8	6	19	174	1296	78
9	6	23	252	7776	150
10	6	59	432	46656	108
11	8	25	336	32768	200
12	8	61	576	262144	144
13	9	16	18	729	135
14	9	22	261	6561	117
15	10	27	420	100000	250
16	10	63	720	1000000	180

Figure 1 Output listings of files WORK.A and WORK.B.

Many people usually use a simple merge to combine the data in two SAS datasets like the code shown below. However, when a dataset has duplicate key values and one file has more records with one key than the other file then the results can turn out to be something that is not accurate, as shown in the following output log.

Output Log 4 – Notes generated from a merge with duplicate by variable values and the merged file.



		i	aa	bb	cc	d	e	f	g
	1	1	11	12	13	.	.	.	.
	2	1	19	23	31	.	.	.	.
	3	2	22	24	26	19	84	32	50
	4	2	38	46	62	55	144	64	36
	5	3	33	36	39	10	6	27	45
	6	3	57	69	93	16	87	81	39
	7	4	44	48	52	21	168	1024	100
	8	4	76	92	124	57	288	4096	72
	9	5	55	60	65	.	.	.	.
	10	5	95	115	155				
1	11	6	66	72	78	13	12	216	90
	12	6	114	138	186	19	174	1296	78
2	13	6	114	138	186	23	252	7776	150
	14	6	114	138	186	59	432	46656	108
	15	7	77	84	91	.	.	.	.
	16	7	133	161	217	.	.	.	.
	17	8	88	96	104	25	336	32768	200
	18	8	152	184	248	61	576	262144	144
	19	9	99	108	117	16	18	729	135
	20	9	171	207	279	22	261	6561	117
	21	10	110	120	130	27	420	100000	250
	22	10	190	230	310	63	720	1000000	180

There are four NOTEs of interest here:

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

This indicates File A has two rows with i=6, File B has four rows with i=6;

NOTE: There were 20 observations read from the data set WORK.A.

1. Variable i has two rows for each number 1 to 10 that generated 20 rows in File A.
   - NOTE: There were 16 observations read from the data set WORK.B.
   - Variable i has two rows for each number where i is a multiple of either 2 or 3 (2, 3, 4, 6, 8, 9, 10), and since 6 is a multiple of both 2 and 3 it has 4 rows. For a total of 16 rows in the file.
2. NOTE: The data set WORK.MERGED has 22 observations and 8 variables.
   - The final output file has 22 rows, 20 where I = a value 1 to 10 and produces two rows for each value from File A, and two rows extra from File B where the value of I was = to 6.
3. Notice also that the variables from File B are missing for rows where I = 1, 5, and 7 since these are not multiples of either 2 or 3.
   - Also notice that rows 13 and 14 have data duplicated from row 12 of File a.

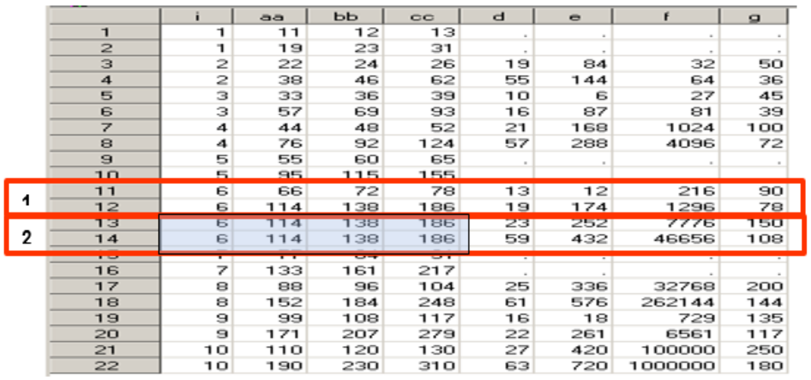
```
1****************** Sample 1 *****************;
2   DATA merged;
3      MERGE a b;
```

```
4      BY i;
5   RUN;

NOTE: MERGE statement has more than one data set with repeats of BY values.
NOTE: There were 20 observations read from the data set WORK.A.
NOTE: There were 16 observations read from the data set WORK.B.
NOTE: The data set WORK.MERGED has 22 observations and 8 variables.
```
…SAS NOTES ABOUT JOB TIMING – SEE ABOVE…
```
6   ***************** Sample 2 *****************;
7   DATA merged;
8      MERGE a b;
9   RUN;

NOTE: There were 20 observations read from the data set WORK.A.
NOTE: There were 16 observations read from the data set WORK.B.
NOTE: The data set WORK.MERGED_2 has 20 observations and 8 variables.
```

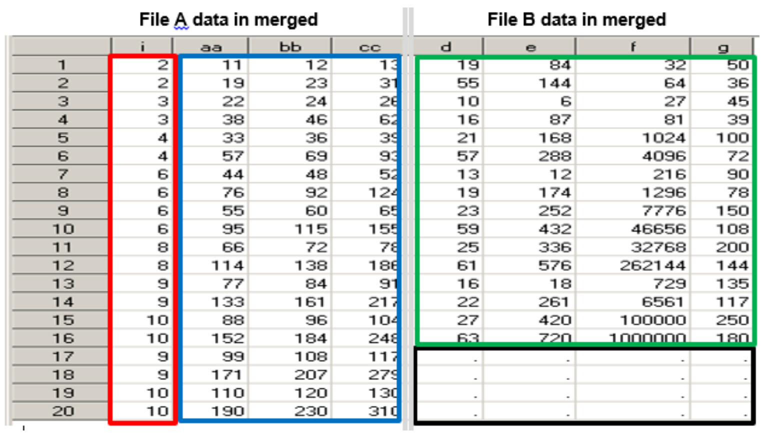
Output Log 4 – Notes generated from a merge with duplicate by variable values and the merged file.



	i	aa	bb	cc	d	e	f	g
1	1	11	12	13	.	.	.	.
2	1	19	23	31	.	.	.	.
3	2	22	24	26	19	84	32	50
4	2	38	46	62	55	144	64	36
5	3	33	36	39	10	6	27	45
6	3	57	69	93	16	87	81	39
7	4	44	48	52	21	168	1024	100
8	4	76	92	124	57	288	4096	72
9	5	55	60	65	.	.	.	.
10	5	95	115	155				
11	6	66	72	78	13	12	216	90
12	6	114	138	186	19	174	1296	78
13	6	114	138	186	23	252	7776	150
14	6	114	138	186	59	432	46656	108
[illegible]	7	[illegible]	[illegible]	[illegible]	.	.	.	.
16	7	133	161	217	.	.	.	.
17	8	88	96	104	25	336	32768	200
18	8	152	184	248	61	576	262144	144
19	9	99	108	117	16	18	729	135
20	9	171	207	279	22	261	6561	117
21	10	110	120	130	27	420	100000	250
22	10	190	230	310	63	720	1000000	180

There are four NOTEs of interest here:

1. NOTE: MERGE statement has more than one data set with repeats of BY values.
   - This indicates File A has two rows with i=6, File B has four rows with i=6;
2. NOTE: There were 20 observations read from the data set WORK.A.
   - Variable i has two rows for each number 1 to 10 that generated 20 rows in File A.
3. NOTE: There were 16 observations read from the data set WORK.B.
   - Variable i has two rows for each number where i is a multiple of either 2 or 3 (2, 3, 4, 6, 8, 9, 10), and since 6 is a multiple of both 2 and 3 it has 4 rows. For a total of 16 rows in the file.
4. NOTE: The data set WORK.MERGED has 22 observations and 8 variables.
   - The final output file has 22 rows, 20 where I = a value 1 to 10 and produces two rows for each value from File A, and two rows extra from File B where the value of I was = to 6.
   - Notice also that the variables from File B are missing for rows where I = 1, 5, and 7 since these are not multiples of either 2 or 3.
   - Also notice that rows 13 and 14 have data duplicated from row 12 of File a.



	File A data in merged				File B data in merged			
	i	aa	bb	cc	d	e	f	g
1	2	11	12	13	19	84	32	50
2	2	19	23	31	55	144	64	36
3	3	22	24	26	10	6	27	45
4	3	38	46	62	16	87	81	39
5	4	33	36	39	21	168	1024	100
6	4	57	69	93	57	288	4096	72
7	6	44	48	52	13	12	216	90
8	6	76	92	124	19	174	1296	78
9	6	55	60	65	23	252	7776	150
10	6	95	115	155	59	432	46656	108
11	8	66	72	78	25	336	32768	200
12	8	114	138	186	61	576	262144	144
13	9	77	84	91	16	18	729	135
14	9	133	161	217	22	261	6561	117
15	10	88	96	104	27	420	100000	250
16	10	152	184	248	63	720	1000000	180
17	9	99	108	117	.	.	.	.
18	9	171	207	279	.	.	.	.
19	10	110	120	130	.	.	.	.
20	10	190	230	310	.	.	.	.

Four squares – **RED** Col I for rows 1-20; **BLUE** Col aa, bb, and cc for rows 1 to 20; **GREEN** Col d, e, f, and g for rows 1 to 16; **BLACK** Col d, e, f, and g for rows 17 to 20.

The messages in Code Sample 2 look like everything worked just fine. But a close examination of the file shows a different result.

1. NOTE: There were 20 observations read from the data set WORK.A.
   - There were 20 rows in file WORK.A.
2. NOTE: There were 16 observations read from the data set WORK.B.
   - There were 16 rows in file WORK.B
3. NOTE: The data set WORK.MERGED_2 has 20 observations and 8 variables.
   - Your expectation might be that only 20 rows would be output.
4. Let's look at the Boxes described above.
   - In The **RED** Box the first 16 values are from WORK.B column i, and the last 4 values are from WORK.A. But the data in the record variables does not match when aligned by site with the key values of variable i.
   - In the **BLUE** Box All data comes from WORK.A because the variables do not exist in WORK.B and WORK.A has 20 rows.
   - In the **GREEN** Box All data comes from WORK.B but it is IN SEQUENTIAL ORDER, not the KEY ORDER suggested by variable i.
   - In the **BLACK** Box we have four rows on missing variables. But when the keys (variable i in both files) are matched WORK.B does not have values of i for 1, 5, and 7. That would indicate 6 rows should be missing.

This indicates that the NOTE (MERGE statement has more than one data set with repeats of BY values.) and the absence of notes on a merge both could indicate you should really look at the output files of a merge step.

RESOLVING MESSAGES

EXAMPLE 4 – MESSAGES THAT HIDE IN PLAIN SIGHT

Now let us examine messages that are similar to ones that we have seen, but have a different root cause. The code below is a log output message, but it is clear enough to show both the SAS code and the log information. First assume that the input file does exist. Cover up the text below line 7, the "RUN" statement and see if you can figure out the problem.

Output Log 5

```
1   LIBNAME xx  "C:\my_sas_files';
2     DATA test;
3        SET xx.shoes;
4        IF region  = 'Asia'    THEN a = 1;
5        IF product = 'Boot'    THEN b = 2;
```

```
6        q = 'help";
NOTE: Library XX does not exist.
7      RUN;
```

Output Log 5 – This NOTE describes a simple syntax problem, can you find it?

Did it take you longer than 30 seconds to find the error? Everything looks great doesn't it? There is a libname statement, a data statement, a set statement, and a run statement. There are also no WARNING, ERROR, or SYNTAX messages. But, look at the quoted strings. The LIBNAME statement quoted string begins with a double quote, and the last line of the program (before the run statement) ends in a double quote. This whole "DATA" step is a badly formed LIBNAME statement.

LIBNAME xx "C:\my_sas_files';

. . .

q = 'help";

EXAMPLE 5 – MESSAGES THAT HIDE IN THE DETAILS

This example has three parts, create dataset "a", create dataset "b", and merge datasets "a" and "b".

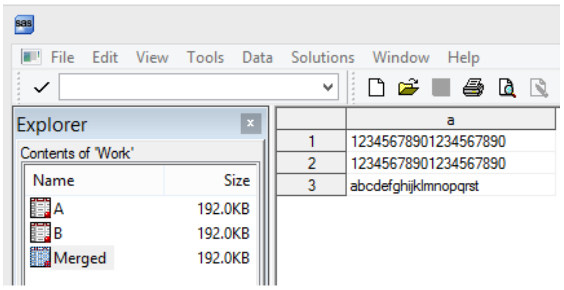
Output Log 6

```
1    DATA a ;
2    a = "12345678901234567890";
3    RUN;
NOTE: The data set WORK.A has 1 observations and 1 variables.
...SAS NOTES ABOUT JOB TIMING – SEE ABOVE...
4    DATA b ;
5    a = "12345678901234567890123456"; output;
6    a = "12345678901234567890765432"; output;
7    a = "abcdefghijklmnopqrstuvwxyz"; output;
8    RUN;
NOTE: The data set WORK.B has 3 observations and 1 variables.
...SAS NOTES ABOUT JOB TIMING – SEE ABOVE...
9     ******************** Sample 1 *******************;
10     DATA merged;
11        MERGE a b;
12        BY a;
13    RUN;
WARNING: Multiple lengths were specified for the BY variable a by input
data sets. This might cause unexpected results.
NOTE: There were 1 observations read from the data set WORK.A.
NOTE: There were 3 observations read from the data set WORK.B.
NOTE: The data set WORK.MERGED has 3 observations and 1 variables.
...SAS NOTES ABOUT JOB TIMING – SEE ABOVE...
14    ******************** Sample 2 *******************;
15    DATA merged;
16        MERGE b a;
17        BY a;
18    RUN;
NOTE: There were 3 observations read from the data set WORK.B.
NOTE: There were 1 observations read from the data set WORK.A.
NOTE: The data set WORK.MERGED has 4 observations and 1 variables.
```

Output Log 6 – A truncation example

Truncation Example - Sample 1

The code looks good, but the message about unexpected results sounds a little scary. A lot of times programmers see this message and ignore it. Most of the time they can. But the example code above is one time that you cannot ignore this message and get away without having an error condition in your output file. This is what the table “MERGED” looks like then the code “MERGE A B;” is executed. The file has three records. But should have four records. The length of variable “A” from file WORK.A is interpreted as 20 bytes. But the data in file WORK.B is 26 bytes long.



Name	Size
A	192.0KB
B	192.0KB
Merged	192.0KB

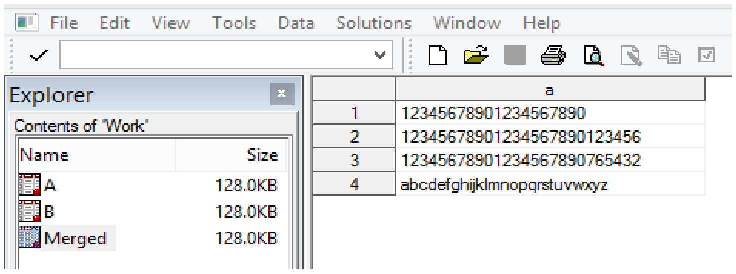
	a
1	12345678901234567890
2	12345678901234567890
3	abcdefghijklmnopqrst

Figure 5 - Result of the merge in output log 6. Sample 1 Merge A B;

The first 20 bytes of variable “a” in file WORK.B are the same as the first 20 bytes on variable “a” in WORK.A and the merge collected all of those records into one output observation.

Truncation Example - Sample 2

Changing the order of the files in the merge statement will make all of the difference in the world. Because the size of the variables is set the first time the variable is encountered in the merge statement. Variable a in WORK.A has a length of 20 bytes, while variable a in WORK.B has a length of 26 bytes. .



Name	Size
A	128.0KB
B	128.0KB
Merged	128.0KB

	a
1	12345678901234567890
2	12345678901234567890123456
3	12345678901234567890765432
4	abcdefghijklmnopqrstuvwxyz

Figure 6 - Result of the merge in output log 6. Sample 2 Merge B A;

This output file has the 4 records that were expected and the length of variable “A” is 26 bytes. (The proportional font makes the letters look like there less than 26 letters.)

SAS ERROR MESSAGE TYPES

Here we will describe different types of error messages.

- Syntax Errors
- Data Errors
- Runtime Errors
- Macro Errors
- Logic Errors

EXAMPLE 6 – SIMPLE SYNTAX ERROR

Output Log 7

```
DATA test;

   IF a=1 THEN b = c;
   IF a2= THEN d = e;
                   -
```

```
                       22
ERROR 22-322: Syntax error, expecting one of the following: !, !!, &, (, *,
**, +, -, /, ;, <, <=, <>, =, >, ><, >=, AND, EQ, GE, GT, IN, LE, LT, MAX,
MIN, NE, NG, NL, NOTIN, OR, [, ^=, {, |, ||, ~=.
5     q = "help";
6
7    RUN;

NOTE: The SAS System stopped processing this step because of errors.
WARNING: The data set WORK.TEST may be incomplete.  When this step was
stopped there were 0 observations and 7 variables.
WARNING: Data set WORK.TEST was not replaced because this step was stopped.
```

Output Log 7 – A syntax error

Notice the underline under the “e” after line 4 and the “22” on the next line. This is where SAS thinks the error occurred. The real error is that the variable in line 4 that is the object of the if statement is “A” not “A2” and the code should read “IF a=2 THEN d = e;”.

EXAMPLE 7 – CASCADING ERRORS ARE NOT ALL ARE DETECTED IN THE FIRST ERROR CHECKING PASS

We will start with this next simple code segment with a statement riddled with errors:

```
DATA out_file_1;
   SET sashelp.shoes (KEEP region sales WHERE(region=asia);
RUN;
```

This example has enough errors that I will show them in several “Passes” I consider this to be an example of “Cascading Errors” because if you fix the first error then others pop up over and over again. (I reset the log line numbers to improve readability, this does not normally occur.)

Pass one

This log message shows some of the errors, the problem is that syntax checking systems present tokens to the checking routines one at a time and some errors will hide others from the checking routines. This code does a good job of hiding errors that you might be able to see. The next log segment will show another missing “=” sign and missing quotes will still be hidden.

Output Log 8

```
1     DATA out_file_1;
2      SET sashelp.shoes (KEEP region sales WHERE(region=asia);
                               ----   ------      ------
                                           12      12
                                                   22
ERROR 12-63: Missing '=' for option KEEP.
ERROR 22-7: Invalid option name REGION.
3     RUN;
NOTE: The SAS System stopped processing this step because of errors.
WARNING: The data set WORK.OUT_FILE_1 may be incomplete.  When this step
was stopped there were 0
         observations and 0 variables.
```

Output Log 8 – First error is a missing “Equals” sign.

The first missing “=” sign caused the DATA step not to execute. Pass two fixes only that condition.

Pass two

This log message does not even seem to know that the word “WHERE” is part of the statement and not a variable name. The real cause is a second missing “=” sign.

Output Log 9

```
1     DATAa out_file_1;
2       SET sashelp.shoes (KEEP=region sales WHERE(region=asia);
                                                   -
                                                   214
                                                   23
                                                    ------
                                                    22
ERROR 214-322: Variable name ( is not valid.
ERROR 23-7: Invalid value for the KEEP option.
ERROR 22-7: Invalid option name REGION.
3     RUN;
NOTE: The SAS System stopped processing this step because of errors.
WARNING: The data set WORK.OUT_FILE_1 may be incomplete.  When this step
was stopped there were 0
         observations and 0 variables.
WARNING: Data set WORK.OUT_FILE_1 was not replaced because this step was
stopped.
```

Output Log 9 – SYNTAX parser gets confused when it finds a Left Parentheses.

Inserting an equal sign after the word WHERE brings us to the next level of error in the statement in the Output Log 10 listing.

Pass Three

This pass finally detects the missing ")" at the end of the set statement but still may not be completely done. One more thing is missing, see if you can detect it now.

Output Log 10

```
1     DATA out_file_1;
2      SET sashelp.shoes (KEEP=region sales WHERE=(region=asia);
                                                              -
                                                              6
                                                              180
ERROR 6-185: Missing ')' parenthesis for data set option list.
ERROR 180-322: Statement is not valid or it is used out of proper order.
3     RUN;
NOTE: The SAS System stopped processing this step because of errors.
WARNING: The data set WORK.OUT_FILE_1 may be incomplete.  When this step
was stopped there were 0 observations and 0 variables.
WARNING: Data set WORK.OUT_FILE_1 was not replaced because this step was
stopped.
```

Output Log 10 – Missing Right Parentheses is finally detected.

Correcting the missing Right Parentheses brings to the next error.

Pass Four

The intent here was to test for a character constant the word "asia". However, the missing quotes around the letters 'asia' caused SAS to look for a variable, not a constant. Had the input dataset had a variable called "asia" then no error would have been identified. The only records selected would have been the ones where the variables "region" and "asia" had exactly the same (Case Sensitive) values. Since there was no variable called "asia" the error was generated. If there was a variable named asia and it was numeric that would have shown another set of errors.

Output Log 11

```
1     DATA out_file_1;
```

```
2      SET sashelp.shoes (KEEP=region sales WHERE=(region=asia));
ERROR: Variable asia is not on file SASHELP.SHOES.
3     RUN;
NOTE: The SAS System stopped processing this step because of errors.
WARNING: The data set WORK.OUT_FILE_1 may be incomplete.  When this step
was stopped there were 0 observations and 0 variables.
WARNING: Data set WORK.OUT_FILE_1 was not replaced because this step was
stopped.
```

Output Log 11 – SAS looked for a variable not a constant.

Pass Five

When the quotes are finale put around the word “asia” we get the following messages, but because the test [region=”asia”] is case sensitive and the data file contains the value “Asia”. No records were selected.

Here this log listing changes from being a real syntax error to a data error. The intent was to test for a Character Constant the word “asia” in the variable REGION. However, the SAS dataset variable called “REGION” does not have a case sensitive value “asia”. No errors were detected and no records were output either.

Output Log 12

```
1     DATA out_file_1;
2      SET sashelp.shoes (KEEP=region sales WHERE=(region="asia"));
3     RUN;

NOTE: There were 0 observations read from the data set SASHELP.SHOES.
      WHERE region='asia';
NOTE: The data set WORK.OUT_FILE_1 has 0 observations and 2 variables.
NOTE: DATA statement used (Total process time):
```

Output Log 12 No SYNTAX Errors and No Records Were Output, A Case sensitive Test Failed.

Pass Six

While you can always code for exactly what you know is in the SAS dataset you can also code for all possible cases by using a SAS function to enhance your test range of input values. The following shows a way to test for all possible cases where the word “aSiA” could be spelled any way in a case sensitive environment.

Output Log 13

```
1     DATA out_file_1;
2      SET sashelp.shoes (KEEP=region sales WHERE=(UPCASE(region)="ASIA"));
3     RUN;
NOTE: There were 14 observations read from the data set SASHELP.SHOES.
      WHERE UPCASE(region)='ASIA';
NOTE: The data set WORK.OUT_FILE_1 has 14 observations and 2 variables.
```

Output Log 13 – Use the UPCASE Function to Find Any Case Sensitive Spelling of Data In the Variable REGION.

EXAMPLE 8 – RUNTIME ERROR MESSAGES

Why are each of the flagged variables listed as uninitialized?

Output Log 14

```
1    DATA test;
2         SET sashelp.shoes;
3         IF z=1 THEN b = c;
4         IF z=2 THEN r = s;
```

```
5         q = "help";
6    RUN;

NOTE: Variable z is uninitialized.
NOTE: Variable c is uninitialized.
NOTE: Variable s is uninitialized.
NOTE: There were 395 observations read from the data set SASHELP.SHOES.
NOTE: The data set WORK.TEST has 395 observations and 13 variables.
```

Output Log 14 – Run-Time Error messages.

The answer is the following:

- None of these variables are on the original input dataset.
- Z is being tested for a value of 1 or 2, but it is not being set to 1 or 2.
- C and S have no values but are being used to place a value into B and R.

EXAMPLE 9 – LOGIC ERRORS

Not many SAS programmers use subroutines, they seem to be a holdover from when a big computer meant the size was big and the memory was small, not the other way around like today. But, Base SAS still retains the power to process subroutines. Subroutines are pieces of code in your program that can be executed many times. (similar to a macro)

Output Log 15

```
1    DATA test;
2       a = 1;     b = 2;
3       LINK add_a_AND_b;
4       IF c > 10 THEN GOTO exit;
5    return;
6    exit:
7    add_a_and_b:
8       c = SUM(a,b);
9       LINK exit;
10   RETURN;
11   RUN;

ERROR: More than 10 LINK statements have been executed at line 9 column 4.
Check your LINK/RETURN logic and use the /STACK= option on the DATA
statement to specify a greater limit for LINK statement nesting.

a=1 b=2 c=3 _ERROR_=1 _N_=1

NOTE: The SAS System stopped processing this step because of errors.
WARNING: The data set WORK.TEST may be incomplete.  When this step was
stopped there were 0 observations and 3 variables.
```

Output Log 15 - Logic Error Caused By Linking to a Subroutine too Many Times.

The problem here is that the values of variables A and B never change, they are reset at the start of each execution of the DATA step, but the step counter _N_ is never changed because lines 4 and 5 are never executed. Line 9 should be removed and the return in line 10 will allow the program to get to steps 4 and 5 and then fail on line 5, the return statement. But similar errors could just make the program run forever.

EXAMPLE 10 – MACRO VARIABLE CODEING ERRORS

This simple looking code segment does not work as it is written. The call to SYMPUTX and the use of the macro variable generated can not be in the same DATA Step.

```
DATA temp;
```

```
        a = 5;
        CALL SYMPUTX('max_array_size1',a,'G');
        ARRAY second(&max_array_size1) _TEMPORARY_ ;
        second(3)=15;
        DO I = 1 TO &max_array_size1;
           PUT second(I)=;
        END;
     RUN;
```

Output Log 16

```
1    DATA temp;
2       a = 5;
3       CALL symputx('max_array_size1',a,'G');
4       ARRAY second(&max_array_size1) _TEMPORARY_ ;
                     -
                     22
                     200
NOTE: The array second has the same name as a SAS-supplied or user-defined
function.  Parentheses following this name are treated as array references
and not function references.

WARNING: Apparent symbolic reference MAX_ARRAY_SIZE1 not resolved.
ERROR 22-322: Syntax error, expecting one of the following: a name, an
integer constant, *.

ERROR 200-322: The symbol is not recognized and will be ignored.

5       second(3)=15;
ERROR: Mixing of implicit and explicit array subscripting is not allowed.
ERROR: Mixing of implicit and explicit array subscripting is not allowed.
6       DO i = 1 TO &max_array_size1;
                    -
                    22
WARNING: Apparent symbolic reference MAX_ARRAY_SIZE1 not resolved.
ERROR 22-322: Syntax error, expecting one of the following: a name, a
quoted string, a numeric constant, a datetime constant, a missing value,
INPUT, PUT.

7       PUT second(I)=;
ERROR: Mixing of implicit and explicit array subscripting is not allowed.
8       END;
9    RUN;

NOTE: The SAS System stopped processing this step because of errors.
WARNING: The data set WORK.TEMP may be incomplete.  When this step was
stopped there were 0 observations and 4 variables.
```

Output Log 16 – Errors generated by Using a SYMPUTX Call to Generate and Use a Macro Variable in the Same DATA Step.

The following output solves the problem, and generates semi-useful/curious messages. See if you can find the changes.

Output Log 17

```
1  DATA _null_;
2     a = 5;
3     CALL SYMPUTX('max_array_size',a,'G');
```

```
4  RUN;
…SAS NOTES ABOUT JOB TIMING – SEE ABOVE…

5    DATA temp;
6       ARRAY second(&max_array_size) _TEMPORARY_ ;
NOTE: The array second has the same name as a SAS-supplied or user-defined
function.  Parentheses following this name are treated as array references
and not function references.
7       second(3)=15;
8       DO i = 1 TO &max_array_size;
9       PUT second(I)=;
10      END;
11   RUN;

NOTE: Compression was disabled for data set WORK.TEMP because compression
overhead would increase the size of the data set.
second[1]=.
second[2]=.
second[3]=15
second[4]=.
second[5]=.
NOTE: The data set WORK.TEMP has 1 observations and 1 variables.
```

Output Log 17 – Corrected usage. This code has a new DATA Step.

EXAMPLE 11 – MACRO SYNTAX ERRORS

See if you can find the issue here, before looking at the Output Log Listing.

```
%LET path = "C:\temp\";
%LET dept = "my_files"";
%MACRO test_macro;
   LIBNAME xx = "&path.&dept.";
   DATA errors;
      SET xx.shoes;
   RUN;
%MEND test_macro;
%test_macro;
RUN;
```

The hint found in these error messages is that the "MEND" statement is not defined. Meaning the "%MACRO test_macro" command was not interpreted correctly. Notice that right above that command is a "%LET" command with mismatched double quote characters. The extra double quote hides the semicolon on line two, all of line three, and the part of line four that is before the first double quote. The result is that the SAS interpreter does not know what you are doing and makes its best guess at how to describe the problem. It does not know how to tell you that you have too many double quotes in line two.

Output Log 18

```
1    %LET path = "C:\temp\";
2    %LET dept = "my_files"";
3    %MACRO test_macro;
ERROR: Open code statement recursion detected.
4       LIBNAME xx = "&path.&dept.";
WARNING: Apparent symbolic reference DEPT not resolved.
5       DATA errors;
6          SET xx.shoes;
7       RUN;
8    %MEND test_macro;
ERROR: Macro keyword MEND appears as text.
```

```
9    %test_macro;
      -
      180
WARNING: Apparent invocation of macro TEST_MACRO not resolved.

ERROR 180-322: Statement is not valid or it is used out of proper order.

10   RUN;
```

Output Log 18 - Line Two has Two Double Quotes.

The problem is that does not fix all of the errors. Now that we have found the extra double quote and removed it from the code we can try again. But it does not seem that we have made much progress because there are still more messages than code. The first error message says we have an "=" sign is the LIBNAME statement, that of course does not belong there. Next we see is that there are way too many double quotes in the generated LIBNAME path (**""C:\temp\""my_files"**) on line 9. We can correct that by removing the double quotes from the %LET statements.

Output Log 19

```
1    %LET path = "C:\temp\";
2    %LET dept = "my_files";
3    %MACRO test_macro;
4       LIBNAME xx = "&path.&dept.";
5       DATA errors;
6          SET xx.shoes;
7       RUN;
8    %MEND test_macro;
9    %test_macro;
ERROR: Libref in LIBNAME statement must be followed either by quoted string
or engine name or semicolon; "=" found.
ERROR: Error in the LIBNAME statement.
NOTE: Line generated by the macro variable "DEPT".
!     ""C:\temp\""my_files"
      --        --
      49        49
ERROR: Libref XX is not assigned.
NOTE: The SAS System stopped processing this step because of errors.
WARNING: The data set WORK.ERRORS may be incomplete.  When this step was
stopped there were 0 observations and 0 variables.

NOTE 49-169: The meaning of an identifier after a quoted string might
change in a future SAS release.  Inserting white space between a quoted
string and the succeeding identifier is recommended.
10   RUN;
```

Output Log 19 – Now LIBNAME Error Messages Appear.

The final correct code produces these messages. Of course this only works if you have a SAS file called "SHOES" in your directory.

Output Log 20

```
1    %LET path = C:\temp\;
2    %LET dept = my_files;
3    %MACRO test_macro;
4       LIBNAME xx "&path.&dept.";
5       DATA errors;
6          SET xx.shoes;
7       RUN;
8    %MEND test_macro;
```

```
9    %test_macro;

NOTE: Libref XX was successfully assigned as follows:
      Engine:        V9
      Physical Name: C:\temp\my_files

NOTE: There were 395 observations read from the data set XX.SHOES.
NOTE: The data set WORK.ERRORS has 395 observations and 7 variables.
10   run;
```

Output Log 20 – Successful Run No Errors.

SUPPRESSING MESSAGES

Some messages can be suppressed but I suggest that you only suppress messages when they would fit into these categories:

- The number of messages would be excessive
- The messages are similar
- The message types have been previously been ex
- The messages do not impact the resulting output

The next question is what type of error messages can be suppressed.

EXAMPLE 12 – SUPPRESS ALL NOTE: MESSAGES

All NOTE messages can be suppressed by using the SAS System Option “NONOTES” as shown here. This is the same code used in the last part of Example 11 and as you can see no NOTES appeared after the code executed. This option remains in effect until you issue an “OPTIONS NOTES” command or end your SAS session. I never recommend using this option, but present it here so you will know it does exist. When you use this SAS option you run the risk of not seeing NOTE messages. These messages are like the ones I have already described that may provide important information about activities occurring within your SAS program. As we have seen some notes are very important when you are trying to figure out why your program is not working as you expect it to work.

Output Log 21

```
1   OPTIONS nonotes;
2   %LET path = C:\temp\;
3   %LET dept = my_files;
4   %MACRO test_macro;
5      LIBNAME xx "&path.&dept.";
6      DATA errors;
7         SET xx.shoes;
8      RUN;
9   %MEND test_macro;
10   %test_macro;
11   RUN;
```

Output Log 21 – All NOTE Messages are Suppressed, Compare with Output Log 20.

EXAMPLE 13 – SUPPRESS MISSING FORMAT ERRORS

The SAS system option “NOFMTERR” will suppress errors that occur when a SAS variable has been assigned a “USER DEFINED” format or informat. The syntax of this option is “OPTION NOFMTERR;”.

EXAMPLE 14 – SUPPRESS DATA CONVERSION ERRORS WHEN USING INPUT STATEMENTS

When you are reading text data with an input command there are two input modifiers that will allow you to suppress messages. The “?” modifier will allow you to suppress input data conversion error messages, and the “??” modifier will allow you to ignore input data conversion messages. This is useful when you are reading a text file that has invalid characters in fields where you expect to find a number. There are many different forms of the input statement so I will just pick one I like to show the syntax of the method to suppress one of these modifiers. The first CARDS record should produce an ERROR message.

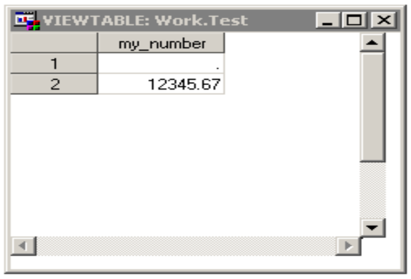
```
DATA test;
INFILE CARDS;
INPUT @1 my_number ?? 8.2;
CARDS;
not_numb
12345.67
;;;;
RUN;
```

Output Log 22

```
1    DATA test;
2    INFILE CARDS;
3    INPUT @1 my_number ?? 8.2;
4    CARDS;

NOTE: The data set WORK.TEST has 2 observations and 1 variables.
7    ;;;;
8    RUN;
```

Output Log 22 – Input Error for variable my_number suppressed when the data is alphanumeric (letters and numbers) not just numbers.



	my_number
1	.
2	12345.67

Figure 7 – Output file WORK.TEST.

EXAMPLE 15 – SUPPRESS MESSAGES WHEN DROPPING, KEEPING, OR RENAMING SAS VARIABLES

This option could have been placed into either the Suppressing or the Generating messages category, but since the default SAS settings cause an error to be generated I placed this example here. SAS has two system options called DKRICOND and DKROCOND. The meaning of the option names really is “DROP” “KEEP” “RENAME” (“INPUT” or “OUTPUT”) “CONDITION”. The function of these options is to enforce how variable names are monitored when you are dropping, keeping, or renaming variables as you either input or output data to SAS datasets. The first listing here is actually the default for most SAS Installations but I chose to show the options for the example. Notice it is real clear that the DATA step did not produce any output. The actual options are “ERROR”, “WARN”, “WARNING”, NOWARN”, and “NOWARNING”.

Output Log 23

```
1  OPTION DKRICOND=ERROR DKROCOND=ERROR;
2    DATA shoes(rename=(smith=jones));
3       SET sashelp.shoes(keep=smith);
ERROR: The variable smith in the DROP, KEEP, or RENAME list has never been
referenced.
4    RUN;

ERROR: The variable smith in the DROP, KEEP, or RENAME list has never been
referenced.
NOTE: The SAS System stopped processing this step because of errors.
```

```
WARNING: The data set WORK.SHOES may be incomplete.  When this step was
stopped there were 0 observations and 0 variables.
```

Output Log 23 – Variable SMITH Not on Input File, so the Error Condition Caused the Job to Fail.

Output Log 24

```
1    OPTION DKRICOND =NOWARN DKROCOND=NOWARNING;
2    DATA shoes(rename=(smith=jones));
3       SET sashelp.shoes(keep=smith);
4    RUN;

NOTE: There were 395 observations read from the data set SASHELP.SHOES.
NOTE: The data set WORK.SHOES has 395 observations and 0 variables.
```

Output Log 24 – Same Code with Options Set to NOWARN Setting. No Warning or Error Messages Were Output.

GENERATING MESSAGES

EXAMPLE 16 – MAKE UP YOUR OWN NOTE, WARNING, AND ERROR MESSAGES

SAS gives you a way to do just about anything you want to do in your programs. Here is an example that allows you to generate your own color coded messages (they picked the colors) to highlight conditions. (all shown here in “Black and White)

Output Log 25

```
1    DATA _null_;
2       PUT 'NOTE       my note       1 - shows up as text on the log';
3       PUT 'WARNING    my warning    1 - shows up as text on the log';
4       PUT 'ERROR      my error      1 - shows up as text on the log';
5    RUN;
NOTE       my note       1 - shows up as text on the log
WARNING    my warning    1 - shows up as text on the log
ERROR      my error      1 - shows up as text on the log
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
6
7    DATA _null_;
8       PUT 'NOTE:      my note       1 - shows up as a NOTE on the log';
9       PUT 'WARNING:   my warning    1 - shows up as a WARNING on the
log';
10      PUT 'ERROR:     my error      1 - shows up as a ERROR on the log';
11   RUN;

NOTE:      my note       1 - shows up as a NOTE on the log
WARNING:   my warning    1 - shows up as a WARNING on the log
ERROR:     my error      1 - shows up as a ERROR on the log
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

Output Log 25 – Ways to Generate Your Own Messages. Notice That the Messages Appear in Different Locations in the Output Log Listing.

Additionally the SAS ERROR statement will allow you to send a message to the SAS Log window, and dump all variable values each time the error occurs (up until the number of errors permitted by the SAS System Option “ERRORS=”. Lines 8, 9,and 10 show up in color on the log.

EXAMPLE 17 – SHOW MORE INFORMATION ABOUT THE RUNTIME ENVIRONMENT

This one last system option shows you information about how long your job ran and what system resources your job used. The actual output may differ on different operating systems but it generally similar. See the log message NOTE: DATA … in Example 16 to review the difference.

Output Log 26

```
1    OPTION FULLSTIMER;
2    DATA shoes;
3       SET sashelp.shoes;
4    RUN;

NOTE: There were 395 observations read from the data set SASHELP.SHOES.
NOTE: The data set WORK.SHOES has 395 observations and 7 variables.
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      user cpu time       0.00 seconds
      system cpu time     0.00 seconds
      memory              526.28k
      OS Memory           12516.00k
      Timestamp           04/09/2014 02:46:46 PM
      Step Count          2  Switch Count  0
```

Output Log 26 – Listing showing Expanded Job Timing Statistics.

CONCLUSION

I hope you have figured out by now that there are no irrelevant message classes within the SAS message system. There are some messages that are truly only informational and could be suppressed without impacting your ability to make your programs run smoother. But even the lowly “NOTE” should not be ignored. If you only search for “ERROR:” when you check your SAS Log output you may still put your job output at risk.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Name:	William E Benjamin Jr
Enterprise:	Owl Computer Consultancy, LLC
Address:	P.O.Box 42434
City, State ZIP:	Phoenix AZ, 85080
Work Phone:	623-337-0269
E-mail:	William@owlcomputerconsultancy.com