Paper TT_14

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 enable you to identify important messages and simple comments about your job and SAS environment and aid you to understand the meaning of a message. This will enable you 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 some experienced SAS users have is determining the meaning of SAS messages on the output log. Some beginners are unable to determine the meaning of messages and spend a lot of time looking for how to determine 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

Of course SAS labels the messages sent to the log as "Error", "Warning", "and "Note". In the next few pages I will explain myself and let you decide for yourself. While both SAS and I put the messages into three categories, I do not always have the messages in the same level as SAS groups them.

Locating Messages

Generally speaking the SAS messages are output to the SAS LOG window when first starting SAS the system 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 to 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. This location can be changed by another PROC PRINTTO command to identify a new file or 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.

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;
```

and this output log listing:

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

NOTE: DATA statement used (Total process time):
      real time           0.20 seconds
      cpu time            0.03 seconds


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]=.
NOTE: The data set WORK.TEMP has 1 observations and 1 variables.
NOTE: DATA statement used (Total process time):
      real time           0.13 seconds
      cpu time            0.00 seconds


10   PROC PRINT DATA=temp;
11   RUN;

NOTE: There were 1 observations read from the data set WORK.TEMP.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.40 seconds
      cpu time            0.04 seconds
```

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

This Note appears when an input file does not exist:

```
27   LILBNAME aa "c:\my_sas_dir";
NOTE: Library AA does not exist.
```

This Note appears when an input file does exist:

```
30   LIBNAME aa "c:\my_sas_code";
NOTE: Libref AA was successfully assigned as follows:
      Engine:        V9
      Physical Name: c:\my_sas_code
```

Not all file actions produce notes

```
31
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 */

36
```

EXAMPLE 3 – MESSAGES THAT RESULT FROM MERGING DATASETS INCORRECTLY

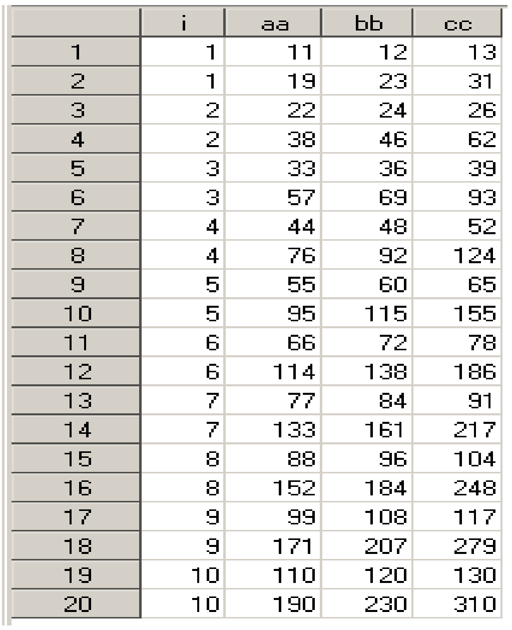
Given this sample code we will look at some different ways to merge SAS datasets. Not all of which generate the correct results. This code produces two files "a" and "b". File a has 20 records and file b has 16 records. Each file has the variable "i" that ranges from 1 to 10. File "a" contains two records 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 for 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;
```

The output log looks like this:

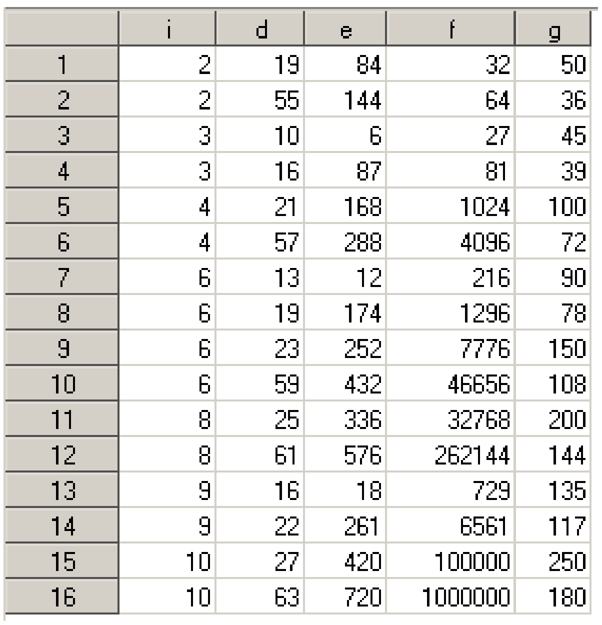
```
     OPTION COMPRESS=binary;
     DATA a(KEEP=i aa bb cc) b(keep=i d e f g);
     DO i = 1 TO 10;
        aa = i*11;    bb = i*12;    cc = i*13;         OUTPUT a;
        aa = i*19;    bb = i*23;    cc = i*31;         OUTPUT a;
        IF MOD(i,3) = 0 THEN DO;
              d = i+7;   e = i*2;   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;

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.
NOTE: DATA statement used (Total process time):
      real time           0.07 seconds
      cpu time            0.03 seconds
```

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

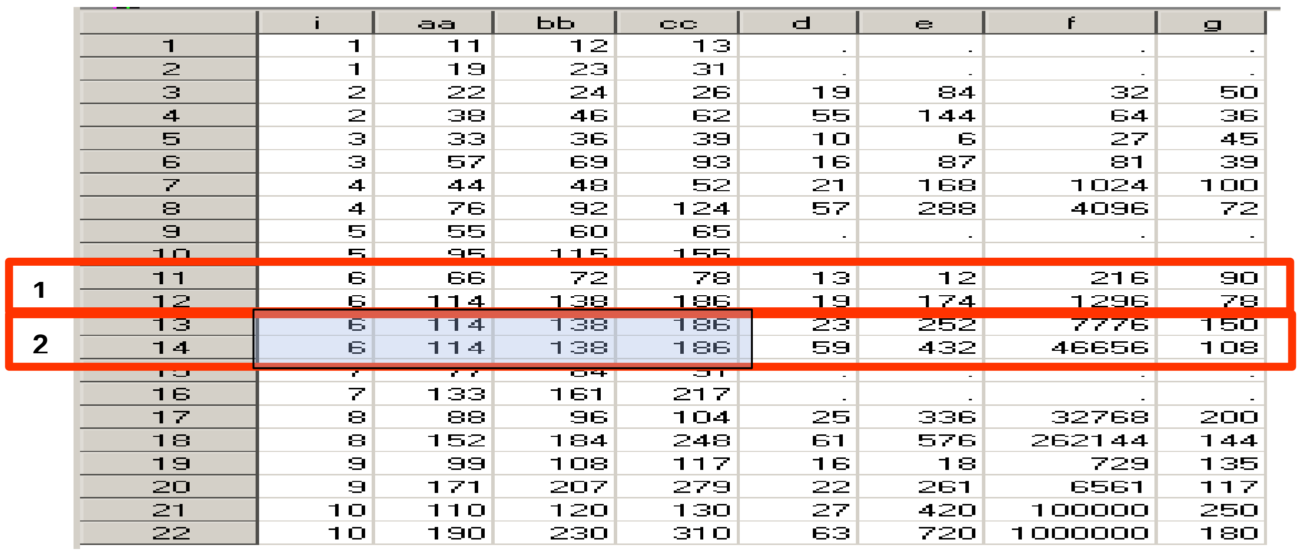
File B



	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

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.

```
DATA merged;
   MERGE a b;
   BY i;
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.
NOTE: DATA statement used (Total process time):
      real time           0.02 seconds
     cpu time           0.01 seconds
```



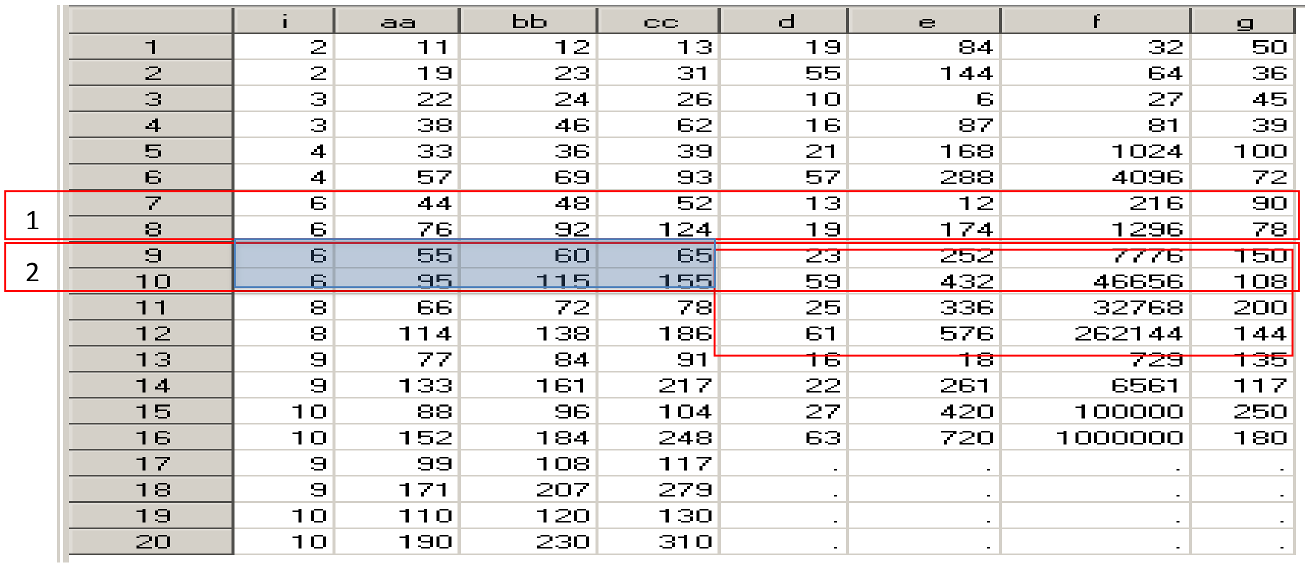
	i	aa	bb	cc	d	e	f	g
1	1	11	12	13	.	.	.	.
2	1	19	23	31	.	.	.	.
3	2	22	24	26	19	84	32	50
4	2	38	46	62	55	144	64	36
5	3	33	36	39	10	6	27	45
6	3	57	69	93	16	87	81	39
7	4	44	48	52	21	168	1024	100
8	4	76	92	124	57	288	4096	72
9	5	55	60	65	.	.	.	.
10	5	95	115	155				
11	6	66	72	78	13	12	216	90
12	6	114	138	186	19	174	1296	78
13	6	114	138	186	23	252	7776	150
14	6	114	138	186	59	432	46656	108
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

Here using a "BY statement "BY I;" shows that the data in rows 13 and 14 have duplicate data that comes from row 12. That is because the number of rows where I = 6 is different in each file. The values in row 13 and 14 (variables aa, bb, and cc) come from the last row of the first file to run out of records for I = 6. This is the standard way that the SAS MERGE statement processes these files when different numbers of key values exist in the files. If this is not what you want you need to adjust the output files.

When you do not use a by statement as in the code below you get different results.

```
DATA merged;
   MERGE a b;
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.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

The messages from this data step with a merge look just fine. There are no strange duplicate key messages and no errors or warnings. The number of input records equals the number of output records. So, from the SAS log this merge looks just fine. But, look at the data file. There are no records for the values of "I" where "I" = 1, 5, or 7. The notes in the SAS log do not tell you the file has been corrupted.



	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	.	.	.	.

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.

```
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;
```

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. 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.

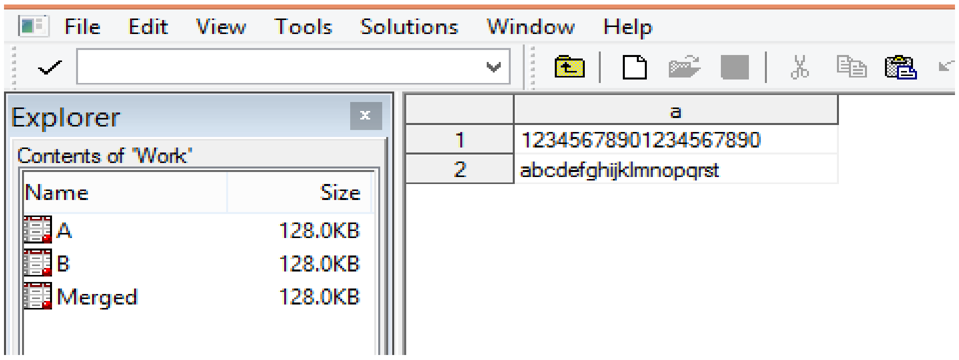
```
LIBNAME xx  "C:\my_sas_files';
. . .
q = 'help";
```

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".

```
1     DATA a ;
2     a = "12345678901234567890";
3     RUN;

NOTE: The data set WORK.A has 1 observations and 1 variables.
NOTE: DATA statement used (Total process time):
      real time           0.04 seconds
      cpu time            0.01 seconds

4     DATA b ;
5     a = "12345678901234567890123456";
6     a = "12345678901234567890765432";
7     a = "abcdefghijklmnopqrstuvwxyz";
8     RUN;

NOTE: The data set WORK.B has 1 observations and 1 variables.
NOTE: DATA statement used (Total process time):
      real time           0.03 seconds
      cpu time            0.00 seconds

9      DATA merged;
10         MERGE a b;
11         BY a;
12     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 1 observations read from the data set WORK.B.
NOTE: The data set WORK.MERGED has 2 observations and 1 variables.
NOTE: DATA statement used (Total process time):
      real time           0.04 seconds
      cpu time            0.00 seconds
```

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 two records. But should have four records. The length of variable "A" from file "A" is used as 20 bytes.

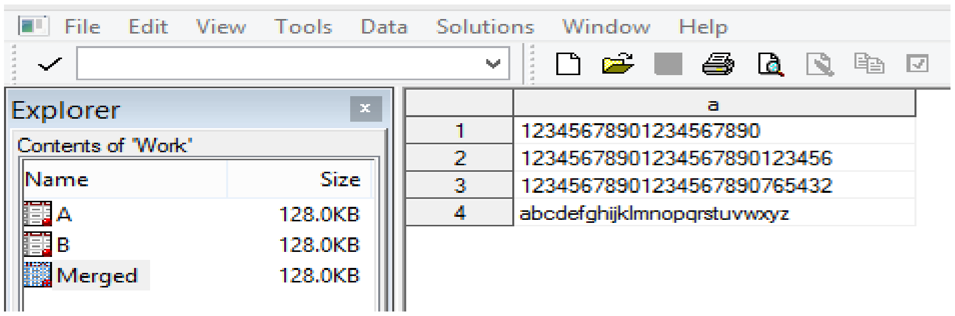


Name	Size
A	128.0KB
B	128.0KB
Merged	128.0KB

	a
1	12345678901234567890
2	abcdefghijklmnopqrst

This is what the table "MERGED" looks like then the code "MERGE B A;" is executed. The file has four records. The length of variable "A" from file "B" is used as 26 bytes. This allowed the merge process to input all of the data.

```
DATA merged;
    MERGE b a;
    BY a;
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.
NOTE: DATA statement used (Total process time):
      real time           0.03 seconds
      cpu time            0.01 seconds
```



Name	Size
A	128.0KB
B	128.0KB
Merged	128.0KB

	a
1	12345678901234567890
2	12345678901234567890123456
3	12345678901234567890765432
4	abcdefghijklmnopqrstuvwxyz

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

The log messages showing the following code presents a syntax error:

```
DATA test;
```

```
2
3      IF a=1 THEN b = c;
4      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.
```

Even though the SAS Error message points to the end of the word "THEN" 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 "A=2" .

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

Next this simple code statement is riddled with errors:

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

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.

```
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.
```

Pass Two – This log message does not even seem to know that the "WHERE" is part of the statement and not a variable name.

```
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.
```

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.

```
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.
```

Pass four – Here this log listing boarders on being either a real syntax error or a logic error. The intent was to test for a Character Constant the word "asia". However if the SAS dataset had a variable called "asia" then no error would have been identified and the only records selected would have been the ones where the variables "region" and "asia" had exactly the same (Case Sensitive) values.

```
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.
```

Pass five – When the quotes are finale put around the word "asia" we get the following messages, because the test [region="asia"] is case sensitive and the data file contains the value "Asia".

```
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):
```

EXAMPLE 8 – DATA ERROR

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 in a case sensitive environment.

```
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.
```

EXAMPLE 9 – RUNTIME ERROR MESSAGES

Why are each of the flagged variables listed as initialized?

```
1     DAATA 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.
```

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 10 – 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)

```
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 bservations and 3 variables.
```

EXAMPLE 11 – MACRO CODE ERRORS

Like the rest of the code in this paper this simple looking code segment does not work as it is written.

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

```
1    DATA temp;
2       a = 5;
3       CALL symputx('max_array_size',a,'G');
4       ARRAY second(&max_array_size) _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_SIZE 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_size;
                    -
                    22
WARNING: Apparent symbolic reference MAX_ARRAY_SIZE 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.
```

All of those errors and warnings can be fixed by changing or adding the three lines of code circled in red here. The call symputx is used to create a macro variable within your SAS code, much like a %let statement. But the macro variable is not created and available to use until "AFTER" the data step ends. By making the code into two data steps it works just fine. But the note about the SAS function stays.

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

See the log below:

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

NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.00 seconds


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.
```

EXAMPLE 12 – MACRO SYNTAX ERRORS

This next piece of code is similar to code that a lot of programmers write but once again in this paper about error messages it does not work.

```
%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. That means that the "%MACRO test_macro" command was not interpreted correctly. We notice that right above that command is a "%LET" command with mismatched double quote characters. That extra double quote (it could have been a single quote) hades 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.

```
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;
```

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.

```
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: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

```
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;
```

Correcting all of those errors results in a successful execution of the macro as shown below.

```
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.
NOTE: DATA statement used (Total process time):
      real time            0.01 seconds
      cpu time            0.01 seconds

10   run;
```

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 examined
- The messages do not impact the resulting output

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

EXAMPLE 13 – 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.

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

```
10    %MEND test_macro;
11    %test_macro;
12    RUN;
```

EXAMPLE 14 – SUPPRESS MISSING FORMAT ERRORS

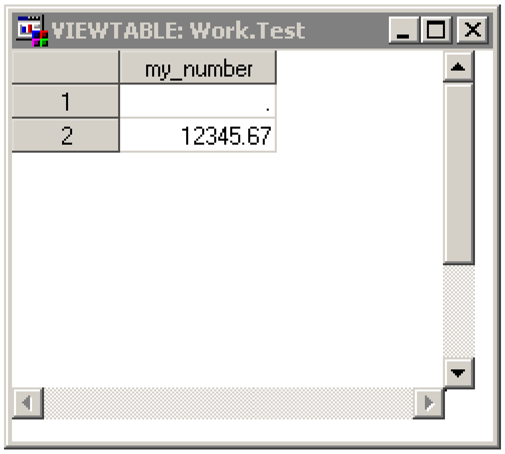
The SAS system option "NOFMTERR" will suppress errors that occur when a SAS variable has been assigned a "USER DEFINED" format or informat. The syntax of this option is "OPTION NOFMTERR;".

EXAMPLE 15 – 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**.**

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

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

NOTE: The data set WORK.TEST has 1 observations and 1 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time           0.01 seconds


7     ;;;;
8     RUN;
```



	my_number
1	.
2	12345.67

Example 16 – 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 variable when 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".

```
OPTION DKRICOND=error DKROCOND=error;

  DATA shoes(rename=(smith=jones));
     SET sashelp.shoes(keep=smith);
ERROR: The variable smith in the DROP, KEEP, or RENAME list has never been
referenced.
  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.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time           0.01 seconds
```

The following is the weakest set of options that can be applied. The NOWARN and NOWARNING do the same thing. It is also a little harder to tell that there are no variables in the output dataset, and therefore no data in the dataset. The log message is a NOTE that tells you exactly how many records the file has each of them with **ZERO** variables. The only variable that would have been kept was called "SMITH" which does not exist in the input file.

```
  OPTION DKRICOND =nowarn DKROCOND=nowarning;

  DATA shoes(rename=(smith=jones));
     SET sashelp.shoes(keep=smith);
  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.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time           0.01 seconds
```

Generating Messages

EXAMPLE 17 – 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

```

DATA _null_;
   PUT 'NOTE      my note       1 - shows up as text on the log';
   PUT 'WARNING   my warning    1 - shows up as text on the log';
   PUT 'ERROR     my error      1 - shows up as text on the log';
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



DATA _null_;
   PUT 'NOTE:      my note       1 - shows up as a NOTE on the log';
   PUT 'WARNING:   my warning    1 - shows up as a WARNING on the log';
   PUT 'ERROR:     my error      1 - shows up as a ERROR on the log';
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
```

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=".

EXAMPLE 18 – SHOW MORE INFORMATION ABOUT THE RUNTIME ENVIRONMENT

One last system option, this 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 17 to review the difference.

```
OPTION FULLSTIMER;

DATA shoes;
   SET sashelp.shoes;
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
```

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