SESUG Paper 182-2019

SAS® PROC IMPORT Troubleshooting Guide

Imelda C. Go and Wendi Wright, Questar Assessment, Inc.

ABSTRACT

Although the task often appears simple, creating data sets from text files with PROC IMPORT can be tricky. This paper takes you through a progression of considerations and complications due to the way PROC IMPORT creates data sets. The discussion focuses on potential problems caused by different factors and, in part, by relying on SAS to define variable attributes; and how to avoid or solve such problems.

When you first started to learn about PROC IMPORT from a book or in a class, you likely had perfect examples. Everything worked as planned and it all looked straightforward. What could go wrong?

PROC IMPORT has a number of options and features that allow you to control how it behaves. The key to surviving its use and minimizing the headaches later is to know how it processes data by default, know what you can do to control/influence how it reads data, and know its limitations. How it behaves depends on the version of SAS and the data.

Due to the breadth of PROC IMPORT's capabilities, only the following selected examples are included. These examples warn of you of a number of complications that you may encounter when using PROC IMPORT to read text files. The code examples were tested with SAS 9.4 TS Level 1M5 with SAS/ACCESS.

ALTERNATIVES TO GIVING PROC IMPORT FULL CONTROL OVER VARIABLE ATTRIBUTES

Any time you do not have full control of how a SAS data set is created, you can run into problems or unexpected difficulties later. Two ways to bypass the uncertainty of SAS-assigned variable attributes are:

1. Create a custom INPUT statement. If you can find the information in the form of a data set, you can create the statement with a macro. This technique is particularly useful for data sets with many variables. If you have control over how these data file layouts are made, you can make them in such a way that it provides the information you need to automate the creation of a custom INPUT statement.
2. Modify or repurpose the INPUT statement generated by PROC IMPORT and that appears in the log. (This is discussed as the last section of this paper.)

STUDY DEFAULT VARIABLE NAMING BEHAVIOR

Let us suppose we have the following in a *test.txt* file:

```
A,B and C,b
1,2,3
a,3,4
```

```
proc import datafile="c:\test.txt" dbms=csv out=test replace;
proc contents;
```

If you do not use the VALIDVARNAME option explicitly, the default or current value will be used. To minimize surprises, making it explicit is a good habit especially if you are already using the option in other programs.

In my SAS environment, the default is VALIDVARNAME=ANY. When I ran the above statement, the following data set resulted:

A	B and C	b
1	2	3
a	3	4

The following appeared among the PROC CONTENTS output.

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format	Informat
1	A	Char	1	$1.	$1.
2	B and C	Num	8	BEST12.	BEST32.
3	b	Num	8	BEST12.	BEST32.

- **Problem: Using the data set in this last example, I am getting error messages that "B and C"n are not valid variable names when the variable is in the data set.**

  If you create a data set when VALIDVARNAME=ANY is in effect, you will need to make sure that option is in effect next time you use that data set.

  If you create a data set with VALIDVARNAME=ANY and attempt to use it when the VALIDVARNAME=V7 is in effect, you will see error messages. In the sample code below, the *test* data set is stored in the DT library after it was created by PROC IMPORT with VARLIDVARNAME=ANY. After the VALIDVARNAME=V7 option takes effect and you attempt to use the data set and the variable, error messages will appear.

```
options validvarname=ANY;
libname dt "c:\";

proc import datafile="c:\test.txt" dbms=csv out=dt.test replace;
proc print data=dt.test; var "b and c"n;

options validvarname=v7;

data sample; set dt.test;
proc print data=dt.test;
var "b and c"n;
run;
```

  The following error messages appeared.

```
58          data sample; set dt.test;

ERROR: The value B and C is not a valid SAS name.
59

NOTE: The SAS System stopped processing this step because of errors.
WARNING: The data set WORK.SAMPLE may be incomplete.  When this step was stopped there
were 0 observations and 0 variables.
WARNING: Data set WORK.SAMPLE was not replaced because this step was stopped.
NOTE: DATA statement used (Total process time):
      real time           0.02 seconds
      cpu time            0.01 seconds

60          proc print data=sample;

61          var "b and c"n;

ERROR: "b and c" is not a valid name.
62          run;

NOTE: The SAS System stopped processing this step because of errors.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

- **Problem: I don't want spaces or special characters in the variable names because it complicates my regular and macro coding. I don't want to type variable names that have spaces in the name since I have to enclose the name in quotes and use a suffix of n (e.g., 'B and C'n).**

  Use the VALIDVARNAME=V7 option so that SAS can turn invalid characters in the variable name to underscores.

```
options validvarname=v7;
proc import datafile="c:\test.txt" dbms=csv out=test replace;
```

A	B_and_C	b
1	2	3
a	3	4

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format	Informat
1	A	Char	1	$1.	$1.
2	B_and_C	Num	8	BEST12.	BEST32.
3	b	Num	8	BEST12.	BEST32.

- **Problem: I don't want to see mixed case variable names because the alphabetic lists of variables from PROC CONTENTS depends on the case of the letters. I want the variable *b* in the example above to appear before variable *B_and_C*.**

  Use the VALIDVARNAME=UPCASE option to activate VALIDVARNAME=V7 and convert all variable name letters to uppercase.

```
options validvarname=UPCASE;
proc import datafile="c:\test.txt" dbms=csv out=test replace;
```

A	B_AND_C	B
1	2	3
a	3	4

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format	Informat
1	A	Char	1	$1.	$1.
3	B	Num	8	BEST12.	BEST32.
2	B_AND_C	Num	8	BEST12.	BEST32.

- **Problem: The variable name was truncated.** The longest SAS variable name is 32 characters long with VALIDVARNAME=V7 and only eight with VALIDVARNAME=V6. If SAS truncates the variable names, you could be left with variable names that are not intuitive or undesirable compared to the full-length variable name.

  Let us suppose we have a *gender.txt* file that looks like this:

```
This is a long variable name just for gender
F
M
```

```
options validvarname=V7;
proc import datafile="c:\gender.txt" dbms=csv out=test replace;
```

  The resulting data set looks like this:

This_is_a_long_variable_name_jus
F
M

- **Problem: An arbitrary variable name was assigned.** If SAS was instructed to get the variable names (GETNAMES=YES is in effect) from the first record in the input file, the values that are not valid SAS names will be assigned default variable names instead. Later, you will likely want to rename these arbitrary names to the correct names.

  Let us suppose we have a CSV file that looks like the following and its first row does not have variable names for the last two variables.

```
*,,
F,B,X
M,W,Y
```

  With VALIDVARNAME=ANY, PROC IMPORT will produce:

*	VAR2	VAR3
F	B	X
M	W	Y

  With VALIDVARNAME=ANY, PROC IMPORT will produce:

_	VAR2	VAR3
F	B	X
M	W	Y

The following information shows the VALIDVARNAME arguments.

(Retrieved from https://documentation.sas.com/?docsetId=acreldb&docsetTarget=n0vnyuzncldjabn1923ug8svx7uh.htm&docsetVersion=9.4&locale=en on August 21, 2019 from the SAS/ACCESS® 9.4 for Relational Databases: Reference, Ninth Edition)

Required Arguments

VALIDVARNAME=V6
indicates that a DBMS column name is changed to a valid SAS name, following these rules:

- Up to eight alphanumeric characters are allowed. Names that are longer than eight characters are truncated. If required, numbers are appended to the ends of the truncated names to make them unique.
- Mixed-case characters are converted to uppercase.
- Special characters are not allowed.

Note: The primary reason for using this value is for existing SAS code that references variables that use older naming rules.

VALIDVARNAME=V7
indicates that a DBMS column name is changed to a valid SAS name, following these rules. This is the default value for SAS 7 and later.

- Up to 32 mixed-case alphanumeric characters are allowed.
- Names must begin with an alphabetic character or an underscore.
- Invalid characters are changed to underscores.
- Any column name that is not unique when it is normalized is made unique by appending a counter (0, 1, 2, and so on) to the name.

VALIDVARNAME=UPCASE
indicates that a DBMS column name is changed to a valid SAS name as described in VALIDVARNAME=V7 except that variable names are in uppercase.

VALIDVARNAME=ANY
allows any characters in DBMS column names to appear as valid characters in SAS variable names. Symbols, such as the equal sign (=) and the asterisk (*), must be contained in a *'variable-name'*n construct. You must use ANY whenever you want to read DBMS column names that do not follow the SAS naming conventions.

Up to 32 characters are allowed in a column name.

Any column name that is not unique when it is normalized is made unique by appending a count (0, 1, 2, and so on).

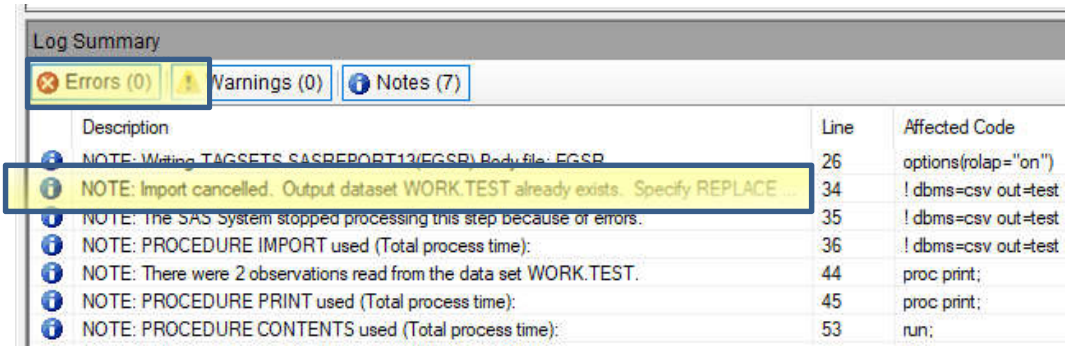
REMEMBER TO USE THE REPLACE OPTION

- **Problem: I was changing the PROC IMPORT statements and rerunning the program, but nothing is changing when it is supposed to.**

  If you do not use the REPLACE option as shown in the code above, you will get a NOTE (and not an ERROR MESSAGE) that the SAS system stopped processing this step because of errors. The following will appear in the SAS log.

```
NOTE: Import cancelled.  Output dataset WORK.TEST already exists.  Specify REPLACE option to overwrite it.
NOTE: The SAS System stopped processing this step because of errors.
```

  If you are using the Log Summary in Enterprise Guide to warn you of errors, this note does not count as an error especially if you meant to replace the data set. The total number of errors is 0.



CHARACTER VS NUMERIC VARIABLE TYPES

Variable types are important because only numeric variable types can be used with procedures that require numeric variables. Unlike the DATA STEP that converts character variable values to numeric ones, PROCs do not make such conversions.

PROC CONTENTS can tell you what the data types are. When looking at PROC PRINT output or data sets in SAS or in SAS Enterprise Guide (EG), the numeric variables are displayed right-justified and the character variables are left-justified.

- **Problem: I have the same variable in two data sets. On one data set it is numeric and on the other it is a character variable.**

  If you attempt to combine two data sets with the same variable that are not of a different type, you will see an error message in the log indicating that the variable has been defined as both character and numeric.

```
ERROR: Variable testvar has been defined as both character and
numeric.
```

- **Problem: If you have a numeric variable that only had blank values, SAS will consider that as a character variable and you cannot use character variables as analysis variables. For example, the following error message appears if you attempt to calculate means on a character variable.**

```
45          proc means;
46          var charvar;
ERROR: Variable charvar in list does not match type prescribed for
this list.
47          run;
```

GUESSINGROWS (ONLY FOR TEXT FILES) CAN BE RIGHT OR WRONG

By default, SAS uses the first 20 records of a file to scan the data and determine the variable type. This default can be changed by increasing the GUESSINGROWS value, which goes from 1 to 2147483647. You may also change the GUESSINGROWS default value by editing the SAS registry.

The option is appropriately named GUESSINGROWS because, at best, SAS can only guess what the variable type should be. The higher the value is, the longer it takes to scan the data.

Given a specific GUESSINGROWS value, a field will become a numeric variable when SAS only sees numbers. It will be a character variable when SAS sees blanks only OR a combination of blanks, numbers, and character values.

If the GUESSINGROWS value is less than the total number of variables in your data file, things could get messy:

- **Problem: Character variable values were truncated.** Character variables may not have the appropriate length and result in truncated values since values of a greater length may appear after the records covered by GUESSINGROWS.
- **Problem: Character variable should have been numeric.** What should be character variables may be numeric because character values appeared after the records covered by GUESSINGROWS.
- **Problem: Data values were lost.** Once the variable type is established as numeric, any values that are not strictly numeric and that appear after the GUESSINGROWS limit are set to missing because they do not convert to valid numeric values. This is a loss of information.

Just because GUESSINGROWS is at least the number of records in the data files does not guarantee the correct character type is chosen.

- **Problem: Numeric variable should have been a character variable.** What if a character variable that has strictly numeric and non-numeric values appears in a data file with only numeric values? That is, the data file may not contain the full range of values possible for a variable. With only numeric values, the variable type will be numeric. You can convert this value to a character variable by using the PUT function as shown in the next example.
- **Problem: My ID values have leading zeroes. Since the data were all numeric, SAS read the data as numbers and all my leading zeroes disappeared.**

  If you want the leading zeroes to appear in the output, you can keep the data type numeric and use the Z format to display the leading zeroes.

  If you want to recover the leading zeroes and convert the numeric values back to character values, you can use the PUT function and the Z format to do this.

  Let us suppose that the ID is 6 digits and that you have the following code.

```
data test;
numvar=343;
charvar=put(numvar,z6.);

proc print;
var numvar charvar;
format numvar z6.;
run;
```

Obs	numvar	charvar
1	000343	000343

- **Problem: Character variable should have been numeric.**

  If you have numeric variables in the file that happened to be blank in the file, they will become character variables. Later, you could run into a problem for those variables (e.g., PROC MEANS analysis variables in the VAR statement must be numeric variables). What you can do is run a check on variables that have to be numbers and convert them into numbers. Here is one way to do it:

`data sample;set sample;`
`if vtype(char)='C' then`
`  do;`
`  charnum=char+0;`
`  drop char;`
`  rename charnum=char;`
`  end;`
`run;`
	The VTYPE function is equal to C when the variable is a character type and is N when the variable is a numeric type.
	
	The score+0 code is to force a conversion on the value in score and scorenum will be a numeric variable.

  You may be tempted to use an array to check on all of these variables, but remember that arrays have to be of the same character type. There are ways to automate this, such as use PROC CONTENTS output to determine which scores should be numbers were rendered as character variables and then run a macro to convert these variables to numeric.

VARIABLE LABELS, FORMATS, AND INFORMATS

In the first example on page 1, we have an example of what variable attributes were assigned by PROC CONTENTS. Note the formats and informats that SAS assigned automatically. The following example does not show labels.

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format	Informat
1	A	Char	1	$1.	$1.
2	B and C	Num	8	BEST12.	BEST32.
3	b	Num	8	BEST12.	BEST32.

If this was a data set read by an INPUT statement, no label, format, or informat will be assigned unless specified by the programmer.

If what SAS does automatically or by default matches your needs there is no issue. How could all of these automatically created things create trouble later? If you need to change an attribute later, you may have to change other attributes for the corresponding variable.

- **Problem: I lengthened a variable using the LENGTH statement in the data step because I had to add new data with longer values. When I look at my SAS output, all new values that are longer appear truncated.**

  Let us supposed you have an ID variable that is of length 11 in a data set created by PROC IMPORT and then you decided to change the length to 20 because you will add data to it that has an ID length greater than 11.

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format	Informat	Label
1	ID	Char	20	$11.	$11.	ID

  However, changing the length did not change the formats and informats. Hence, if you forgot that the format was set to display up to 11 characters only, you will be wondering why you cannot see beyond the 11th character. You will need to change the format to $20. If you want to see the full 20 characters possible.

- **Problem: I do not need all those labels, formats, and informats PROC IMPORT automatically assigned.**

  If you prefer, you can just delete all of these SAS-assigned labels, formats, and informats. This way you can control these attributes yourself and/or get no interference later as you change these attributes. You can remove these attributes from the *sample* data set in the *work* library using the following code.

```
proc datasets library=work;
   modify sample;
    attrib _all_ label='';
    format _all_;
    informat _all_;
quit;
```

DATA SETS WITH THE SAME VARIABLE NAMES BUT DIFFERENT ATTRIBUTES

Let us suppose you have two data sets with the exact same variable names and which were created by PROC IMPORT from two different data files.

- **Problem: I tried to combine the two data sets together with the SET statement and I can't combine them because some variables are not of the same type.**

  When combining data sets, variables with the same name must have the same variable type. Otherwise, an error will result.

- **Problem: I tried to combine two data sets together with the SET statement and the contents of the same variable in the second data set got truncated.**

  Within the GUESSINGROWS limit, PROC IMPORT determined the variable length based on the maximum length of the values it encountered. Based on the range of values encountered, each data set may have a different length.

  In a SET statement, character variable length is determined based on the length of the first time the variable occurs among the data sets listed in a SET statement. Based on the order the data sets are listed in the SET statement, the values got truncated because the length of the variable in the second data set is less than the length of the same variable in the first data set.

DATA INTERFERES WITH DELIMITERS

- **Problem: I needed to use a comma delimiter for a file, but the data itself has commas, which are interfering with how SAS creates the data set.**

  We might receive files that were not tested. Here are a few options that involve asking for a new data file. Of course, the option may not be feasible depending on the nature of the data.

  - Use another delimiter that will not conflict with the data.
  - If the variable affected is a character variable, consider using text delimiters (e.g., enclose character variables with double quotes). (Note: If the data also contain double quotes that could interfere with the double quotes used to delimit the text values.)
  - Remove the delimiter from all the values in the file so that there will be no interference.
  - Replace the delimiter with another value in the file so that there will be no interference.

WHAT YOU CANNOT SEE CAN HURT YOU

- **Problem: The text data file has thousands of records, but SAS only reads 200+. I looked at the data in the text file and saw nothing wrong with the data.**

  The following is the same sample data file from page1 but with a hexadecimal character on the second line after the one and the comma. This time the file is named “sample with hex.txt” and we will use PROC IMPORT to read it.

  What you see depends on your editor (specific version and capabilities). In Notepad, it displayed where I have highlighted in yellow what would be the hexadecimal character.

```
A,B and C,b
1,→2,3
a,3,4
```

  In the NOTEPAD++ editor, I see the 1A hexadecimal value which appears as SUB:

```
A,B and C,b
1,SUB2,3
a,3,4
```

```
proc import datafile="c:\sample with hex.txt" dbms=csv out=test replace;
getnames=yes;
```

  The log has the following:

```
NOTE: The infile 'c:\sample with hex.txt' is:
      Filename=c:\sample with hex.txt,
      RECFM=V,LRECL=32767,File Size (bytes)=28,
      Last Modified=20Aug2019:12:00:23,
      Create Time=20Aug2019:11:52:44

NOTE: 1 record was read from the infile 'c:\sample with hex.txt'.
      The minimum record length was 2.
      The maximum record length was 2.
NOTE: The data set WORK.TEST has 1 observations and 2 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds


1 rows created in WORK.TEST from c:\sample with hex.txt.
```

  We know that the *test* data set should have two records. Note that the log says it only read one record. This is happening because of the hexadecimal value. In Windows, the hexadecimal character 1A stops the processing. Anything past that character on the same record and subsequent records is ignored and not processed. This is a legacy of the DOS days when the hexadecimal 1A was an end-of-file (EOF) character.

  Using the IGNOREDOSEOF SAS option can force SAS to ignore the 1A as a DOS end-of-file character. But you can use this with the FILENAME or INFILE statement only and not with PROC IMPORT. To fix this, you may have to find the troublesome characters using an editor capable of letting you see them or write a program that will create the exact same file without the troublesome characters and then read in the file again using PROC IMPORT.

- **Problem: How do I remove non-printable and special characters from my data?**

  The following lists common examples of nonprintable characters that has the potential to interfere with PROC IMPORT data processing and, as I have seen it, other PROCs that process character variables.

  - Horizontal Tab (HT, 9 decimal value)
  - Line Feed (LF, 10 decimal value)
  - Carriage Return (CR, 13 decimal value)

  The COMPRESS function is useful for removing characters from a variable. For example, the following statement removes the horizontal tab character.

  ```
  newvar = compress(var,'09'x);
  ```

  - If you need the characters, you can replace them (use the TRANSLATE function).
  - If you do not need the characters, you can delete them.
  - Use the following to remove all non-writable characters.

  ```
  charvar= compress(charvar, , 'kw');
  ```

  K stands for Keep and W stands for writable (i.e., printable). There is no second argument to this function call. This statement will keep only the writable characters in the character variable (i.e., it will remove all non-writable characters).

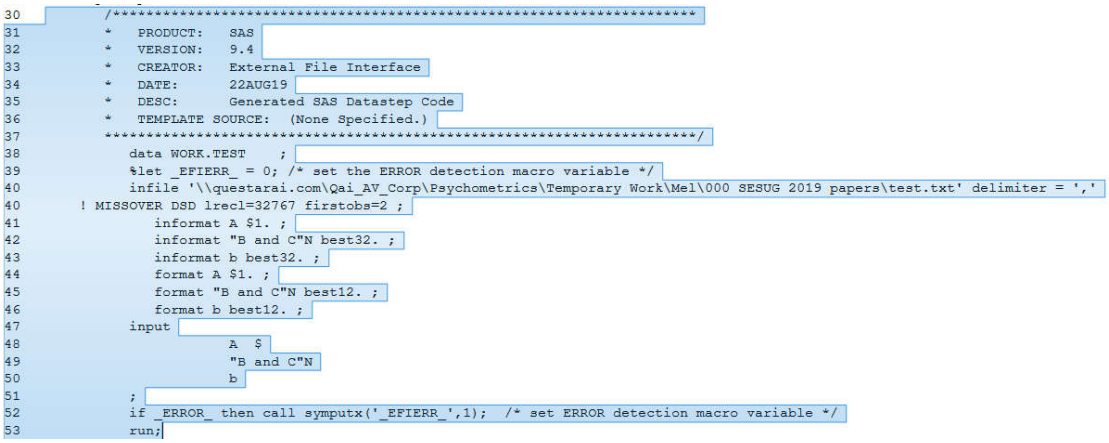
REPURPOSING THE CODE SAS GENERATED FOR PROC IMPORT

After you invoke PROC IMPORT, you may take the statements and make modifications to suit your needs. The following is an example where VALIDVARNAMES=ANY was in effect.

```
30      /*---------------------------------------------------------------------*
31       *   PRODUCT:   SAS
32       *   VERSION:   9.4
33       *   CREATOR:   External File Interface
34       *   DATE:      22AUG19
35       *   DESC:      Generated SAS Datastep Code
36       *   TEMPLATE SOURCE:  (None Specified.)
37       *---------------------------------------------------------------------*/
38          data WORK.TEST    ;
39          %let _EFIERR_ = 0; /* set the ERROR detection macro variable */
40          infile '\\questarai.com\Qai_AV_Corp\Psychometrics\Temporary Work\Mel\000 SESUG 2019 papers\test.txt' delimiter = ','
40        ! MISSOVER DSD lrecl=32767 firstobs=2 ;
41             informat A $1. ;
42             informat "B and C"N best32. ;
43             informat b best32. ;
44             format A $1. ;
45             format "B and C"N best12. ;
46             format b best12. ;
47          input
48                      A  $
49                      "B and C"N
50                      b
51          ;
52          if _ERROR_ then call symputx('_EFIERR_',1);  /* set ERROR detection macro variable */
53          run;
```

I use the SAS Enterprise Guide editor. I press the ALT key while selecting the text in the log. This allows me to select a block of text and effectively exclude the line numbers from the selection. I can then edit this to my satisfaction.

This is how it looks when you select the text without pressing the ALT key.



```
30          /*************************************************************************
31          *   PRODUCT:   SAS
32          *   VERSION:   9.4
33          *   CREATOR:   External File Interface
34          *   DATE:      22AUG19
35          *   DESC:      Generated SAS Datastep Code
36          *   TEMPLATE SOURCE:  (None Specified.)
37          *************************************************************************/
38             data WORK.TEST    ;
39             %let _EFIERR_ = 0; /* set the ERROR detection macro variable */
40             infile '\\questarai.com\Qai_AV_Corp\Psychometrics\Temporary Work\Mel\000 SESUG 2019 papers\test.txt' delimiter = ','
40       ! MISSOVER DSD lrecl=32767 firstobs=2 ;
41                informat A $1. ;
42                informat "B and C"N best32. ;
43                informat b best32. ;
44                format A $1. ;
45                format "B and C"N best12. ;
46                format b best12. ;
47             input
48                         A  $
49                         "B and C"N
50                         b
51             ;
52             if _ERROR_ then call symputx('_EFIERR_',1);  /* set ERROR detection macro variable */
53             run;
```

Another way is to copy the text from the log and do what you need to outside of SAS.

CONCLUSION

SAS provides a number of ways to create SAS data sets. An understanding of how SAS processes data and how things can go wrong during the data-set-creation process is essential to anticipating problems that need to be addressed. Being unaware of how problems can occur could potentially allow such problems to occur undetected, and thus complicate the data processing and endanger the integrity of the data.

CONTACT INFORMATION

Imelda C. Go, Ph.D.
igo@questarai.com
Working remotely from Columbia, SC

Wendi Wright
(717) 513-0027
wrightwendi6@gmail.com
1351 Fishing Creek Valley Rd.
Harrisburg, PA 17112

TRADEMARK NOTICE