Frequently Asked Questions (FAQs) About SAS Usage

Editorial Note: Many of these FAQs are provided courtesy of the University of Texas Statistical Support Center. Although there are references in this document to SAS Version 8.2, the answers given are equally applicable to SAS Version 9.1.3.

TABLE OF CONTENTS

CATCHING DATA ENTRY ERRORS WITH SAS 2
REMOVING DUPLICATE OBSERVATIONS FROM A DATASET USING SAS 2
SAS MISSING VALUES 3
MEAN SUBSTITUTION FOR MISSING VALUES IN SAS 4
RECODING VARIABLE VALUES INTO MISSING VALUES IN SAS 5
READING MISSING DATA USING SAS 6
REPLACING MISSING DATA FROM A SECOND FILE 8
IDENTIFYING NONMATCHES IN A SAS MATCH MERGE 9
SUMMING VARIABLES WITH MISSING DATA IN SAS 10
EXTRACTING CASES WITH A GIVEN STRING FROM SAS 11
USING AN ARRAY IN SAS TO DETECT MISSING VALUES 12
ONE-TO-MANY DATASET MERGING WITH SAS 13
USING THE SAS KEEP & DROP STATEMENTS 14
COMBINING MULTIPLE LINES INTO ONE USING SAS 15
FORMATTING SAS DATA 16
CREATING A SAS DATA SET FROM AN ASCII FILE 17
FILE RELATIONSHIPS IN SAS 20
BIBLIOGRAPHY 22

CATCHING DATA ENTRY ERRORS WITH SAS

Question:
How can I use SAS to check for data entry errors?

Answer:
PROC COMPARE compares two SAS datasets with each other. It warns you if it detects observations (rows) or variables (columns) that do not agree across the two datasets. When there are no disagreements, you can be confident that data entry is reliable. To use **PROC COMPARE**, enter your data twice, once each into two separate raw data files. Then use the two raw data files to create two SAS data sets. Then use **PROC COMPARE**. The following example compares the two SAS data sets named FRED and SAM.

```
PROC COMPARE BASE = fred COMPARE = sam ERROR ;
ID subjctid ;
```

The BASE keyword defines the data set that SAS will use as a basis for comparison. The keyword COMPARE defines the dataset which SAS will compare with the base dataset. The ERROR keyword requests that SAS print an error message to the SASLOG file if it discovers any differences when it compares the two data sets.

The ID statement tells SAS to compare rows (observations) in the data set by the identifying variable, which here is named SUBJCTID. This variable must have a unique value for each case.

PROC COMPARE features a number of options, many of which are designed to control the amount and type of information displayed in the listing file. More information about PROC COMPARE is available in the SAS Procedures Guide.

REMOVING DUPLICATE OBSERVATIONS FROM A DATASET USING SAS

Question:
How can I remove duplication observations from my SAS dataset?

Answer:
You can use **PROC SORT** with the **NODUPLICATES** option to remove unwanted duplicate observations from your SAS dataset. The following sample code illustrates how to use **PROC SORT** to do this.

```
DATA test ;
INPUT id varone vartwo ;
CARDS ;
1 23 45
2 35 98
```

```
3 83 45
1 23 45
;
PROC PRINT ;
PROC SORT IN=test OUT=test2 NODUPLICATES ;
BY _ALL_;
PROC PRINT DATA=test2 ;
RUN ;
```

The **_ALL_** keyword is required for SAS to correctly identify and remove the duplicate observations.

If you want to remove duplicate records for specific BY variables *only* (as opposed to all variables, shown above) substitute the keyword **NODUPKEY** for the **NODUPLICATES** keyword in the example above. In addition, you should substitute a specific BY variable (e.g., vartwo) for the **_ALL_** keyword used in the above example.

For more information, refer to the SAS Procedures Guide, Version 8, Third Edition.

SAS MISSING VALUES

Question:
How does SAS deal with missing data?

Answer:
Whenever SAS encounters an invalid or blank value in the file being read, the value is defined as missing. In all subsequent processes and output, the value is represented as a period (if the variable is numeric-valued) or is left blank (if the variable is character-valued).

In DATA step programming, use a period to refer to missing numeric values. For example, to recode missing values in the variable A to the value 99, use the following statement:

```
IF a=. THEN a=99;
```

Use the **MISSING** statement to define certain characters to represent special missing values for all numeric variables. The special missing values can be any of the 26 letters of the alphabet, or an underscore. In the example below, the values 'a' and 'b' will be interpreted as special missing values for every numeric variable.

```
MISSING a b ;
```

For more information, refer to the ***SAS Language Guide: Reference***,Version 8, First Edition, Chapter 9, the MISSING section.

Mean substitution for missing values in SAS

Question:
How can I replace missing values with a mean value in SAS? Is there an easy way to do this for many variables?

Answer:

SAS has a procedure called **PROC STANDARD** that can be used to standardize some or all of the variables in a SAS data set to a given mean and/or standard deviation and produce a new SAS data set that contains the standardized values. In addition, there is a **REPLACE** option that substitutes all missing values with the variable mean. If the **MEAN=*mean-value*** option is also specified, missing values are set instead to the user-specified ***mean-value***.

The following SAS code demonstrates the use of **PROC STANDARD** for mean substitution.

```
DATA raw ;
  INPUT v1-v10 ;
CARDS;
1 1 1 1 1 . 1 1 1 1
2 2 2 . 2 . 2 2 2 2
3 3 3 3 3 3 . . 3 3
4 4 4 . . 4 4 4 4 4
5 5 5 5 5 5 5 5 . .
;

PROC STANDARD DATA=raw OUT=stnd REPLACE PRINT;
  VAR v1-v10;
RUN;
```

The following SAS code demonstrates another way of substituting mean values for missing values.

```
DATA raw ;
  INPUT v1-v10;
CARDS;
1 1 1 1 1 . 1 1 1 1
2 2 2 . 2 . 2 2 2 2
3 3 3 3 3 3 . . 3 3
4 4 4 . . 4 4 4 4 4
5 5 5 5 5 5 5 5 . .
;
```

```
PROC MEANS NOPRINT;
  VAR v1-v10;
  OUTPUT OUT=meandat(DROP=_TYPE_ _FREQ_) MEAN=m1-m10;
RUN ;

PROC PRINT DATA=meandat;
RUN ;

DATA meansub (DROP=m1-m10 i);
  IF _N_ = 1 THEN SET meandat;
  SET raw;
  ARRAY old(10) v1-v10;
  ARRAY means(10) m1-m10;
  DO i = 1 TO 10;
    IF old(i) EQ . THEN old(i) = means(i);
  END;
RUN;
```

In the first DATA step, the data set **raw** is created with 10 variables, **v1** through **v10**. Notice that there are one or more missing values (periods) for each observation in the data records.

PROC MEANS is used to produce a new dataset **meandat** which has variables **m1** through **m10** holding the means for the variables **v1** through **v10**. **PROC PRINT** is used to verify this.

The second DATA step performs the substitution, creating a final data set called **meansub**. It defines two arrays: **old** represents **v1** through **v10** and **means** represents **m1** through **m10**. A DO loop moves through the array variables, checking each value of array **old** to see if it is missing. If it is missing, then the value is set to the corresponding value from the array **means**; this is the mean substitution.

Recoding Variable Values into Missing Values in SAS

Question:
I would like to recode numeric zeros in my SAS dataset into SAS system missing values. How can I perform this operation?

Answer:
You can accomplish this task by using an **IF-THEN** statement in SAS. SAS uses the period symbol ('.') as its missing value identifier.

The following example shows how to convert zeros to the SAS system missing value code.

```
* SAS Program converts zero numeric values to SAS system missing values ;

DATA ;
INFILE CARDS ;
INPUT id a b c ;
IF a = 0 THEN a = . ;
IF b = 0 THEN b = . ;

CARDS ;
1 2 3 4
2 0 4 5
3 4 0 6
;

PROC PRINT ;
TITLE ' Check for program accuracy' ;
RUN ;

* End sample program ;
```

For more information on recoding numeric values into missing data values, please consult the SAS Language Guide, Version 8, First Edition.

READING MISSING DATA USING SAS

Question:
The option MISSOVER does not work when the missing values are not at the end of the observation, correct?

To read this, for example:

```
546               8456
         111               5555
```

I did:

```
OPTIONS linesize=80 REPLACE NODATE;

FILENAME indata '/home/grad/veronica/LESSONS/SAS/miss2.txt';

DATA NEW1;
   INFILE indata missover;
   INPUT @1 year @9 foodcons @17 prretail @25 dispinc;
```

```
RUN ;

PROC PRINT;
RUN;
```

and got:

```
The SAS System
1

OBS YEAR FOODCONS PRRETAIL DISPINC

1       546     8456    8456     .
2       111     111     5555     5555
```

Is there a way for SAS to interpret empty positions in the middle of a record as missing values?

Answer:
One way to fix the problem is to explicitly define the column width of each variable using SAS formatting input.

The modified program looks like this:

```
OPTIONS LS=80 REPLACE NODATE ;

DATA new1 ;
   INFILE CARDS MISSOVER ;

   INPUT @1 (year) (3.) @9 (foodcons) (3.) @ 17 (pretail) (4.) @25
(dispinc) (4.) ;
CARDS ;
546              8456
         111              5555
;
RUN ;

PROC PRINT ;
RUN ;
```

The (3.) and (4.) in the **INPUT** statement refer to the column width of each variable in question: 3 columns and 4 columns, respectively. Note that if you had data with columns appearing behind a decimal point, such as a GPA of 3.46, then your column specification would be 4.2: 4 total columns, including the decimal point, with two of those four columns consisting of values behind (to the right of) the decimal point.

The program results in the following output:

```
The SAS System 1

OBS YEAR FOODCONS PRETAIL DISPINC
```

```
1       546     .       8456    .
2       .       111     .       5555
```

REPLACING MISSING DATA FROM A SECOND FILE

Question:
I have one data file with some missing values in it. I want to replace those with values from a second data file, but only replace those with matching observation values in the second data file.

Answer:
The following SAS program will replace missing data in dataset one with a matching observation from dataset two.

```
DATA one ;
   INPUT x y z $ ;
CARDS ;
1 2 a1
1 3 b1
1 . c1
2 5 d1
4 6 e1
;

DATA two ;
N+1 ;
INPUT y z2 $ ;
CARDS ;
2 a1
3 b1
4 c1
5 d1
6 e1
;

DATA both ;
   SET one ;
N = .;
IF y=. THEN DO WHILE (z2 NE z) ;
  SET two ;
   END;
DROP z2 ;
PROC PRINT ;
RUN ;
```

IDENTIFYING NONMATCHES IN A SAS MATCH MERGE

Question:
I am match-merging two SAS data sets. I would like to be able to identify, remove, and print out any cases that don't get a match. How can I do this?

Answer:
Use the **IN=** option in the **MERGE** statement to identify the observations that have data from both data sets. The following code demonstrates this:

```
DATA one ;
   INPUT x y z ;
CARDS ;
2 2 3
4 5 6
7 8 9
;
RUN;

DATA two ;
   INPUT x y z ;
CARDS ;
1 2 3
4 5 6
7 8 9
;
RUN;

PROC SORT DATA=one;
   BY x ;
RUN;

PROC SORT DATA=two;
   BY x ;
RUN;

DATA  mergdset  missgset ;
   MERGE one (IN = fromone)  two (IN = fromtwo) ;
   BY x ;
   IF fromone = 1 AND fromtwo = 1 THEN OUTPUT mergdset ;
   ELSE OUTPUT missgset ;
RUN;

PROC PRINT DATA = mergdset ;
   TITLE ' Matched and Merged Observations' ;

PROC PRINT DATA = missgset ;
   TITLE ' Unmatched Observations' ;
RUN ;
```

In this example, the two SAS data sets ONE and TWO are sorted and merged by the variable X. The **IN=** options creates two new variables that will equal 1 if the corresponding data set contributed data to the current observation, or 0 otherwise. The **IF/ELSE** statements then use these variables to output any unmatched observations to the SAS data set MISSGSET, and all other observations to the data set MERGDSET.

More information about the IN= option is available in SAS Language: Reference, Version 8, First Edition.

SUMMING VARIABLES WITH MISSING DATA IN SAS

Question:
I have a number of variables I need to add together using SAS. Some of them have missing data. What is the best way for me to add them together given that I have missing data?

Answer:
There are two general approaches you can take summing variables in SAS.

1. The direct adding method.
With this method, you compute a new variable as a straight sum of the current variables you wish to add.

Newvar = Oldvar1 + Oldvar2 + Oldvar3 ;
In this example, Newvar is the sum of Oldvar1 + Oldvar2 + Oldvar3. If Oldvar1 *or* Oldvar2 *or* Oldvar3 had missing data for a given case, then the value of Newvar for that case would also be missing. In other words, if any of the variables to be summed have missing data, the new variable will also have missing data.

2. The function method.
With the function method, you use the SAS **SUM (OF** operator to add up a number of variables. The advantage of this method is that the syntax is much less laborious to type, especially for large numbers of variables.

Newvar = SUM (OF Oldvar1-Oldvar3) ;
Unfortunately, with this method any variable to be summed which has a missing value is treated as zero by SAS. This means, for example, that if Oldvar1 had a value of 4 and Oldvar2 was missing and Oldvar3 had a value of 3, the value of Newvar would be 7 when the **SUM (OF** function is used. By contrast, the value of Newvar would be missing under method (1) described previously (where you add the variables together using a plus sign).

If you have both a large number of variables to sum and missing data, what can you do? One solution (provided by Karl Wuensch over the Internet) is use the **NMISS (OF** function in conjunction with the **SUM (OF** function, like so:

IF NMISS(OF Oldvar1-Oldvar3) > 0 then Newvar = . ;
ELSE Newvar = SUM(OF Oldvar1-Oldvar3) ;
This code first calculates the number of missing values across the variables Oldvar1 through Oldvar3. If SAS finds any missing data,
it sets the value of Newvar to be missing. Otherwise, the value of Newvar is set to be the sum of the Oldvar1 through Oldvar3 values which have non-missing cases.

EXTRACTING CASES WITH A GIVEN STRING FROM SAS

Question:
I am trying to extract data on a certain drug from a very large database. A frequency tabulation reveals that this drug is coded/spelled in a variety of ways. All of the codes, however, contain the word BACTRIM --e.g. BACTRIM D.S. or BACTRIM D or DS1/ BACTRIM.

Is there an efficient way to select all of these values with one subsetting if statement without having to write out each value? It is written a total of 50 ways. Can I use a substring function? If so, how? Could you provide me with a bit of sample code for doing this?

Answer:
The sample code shown below should help you to configure your SAS program to extract only the records containing "BACTRIM".

```
DATA one ;
 LENGTH drug $20 ;
   INFILE CARDS ;
      INPUT DRUG & ;
CARDS ;
BACTRIM D.S.
BACTRIM D
DS1/ BACTRIM
NOT VALID
ANOTHER NOT VALID
;
RUN ;

DATA two ;
  SET one ;
  flag = 'BACTRIM';
  new_drug = INDEX(drug,flag);
RUN ;

PROC PRINT DATA = two ;
  TITLE 'All records';
RUN ;
```

```
DATA three ;
   SET two ;
   IF new_drug NE 0 ;
RUN ;

PROC PRINT DATA = three ;
  TITLE 'Reduced database';
RUN ;
```

The first DATA step reads in the sample records you sent in your E-mail message, plus a couple of additional records that do not meet the BACTRIM criterion.

The second DATA step gets the information from the first DATA step using the **SET** statement. Then we create a new variable called "flag". Flag is the string we use to limit the records for inclusion. In this case it is "BACTRIM". Now we create another new variable called "new_drug". New_drug is actually a numeric variable. SAS has a function called **INDEX** that will return the location number of the first character in the string. The first variable required by **INDEX** is the variable of interest. In this case that is the variable "drug". The second variable required by **INDEX** is the text string to search for; in this case that is the variable "flag".

The third DATA step gets the information from the second DATA step and limits the cases chosen for inclusion in the third DATA step to only those that have a new_drug number not equal to zero. The **INDEX** function returns a zero value for new_drug if it cannot find the string "BACTRIM" in the subject's drug variable, so we want to include only cases that have a non-zero value for new_drug.

It is possible and even desirable to compress this program into a single DATA step. For purposes of illustration we show each step in clear and distinct parts.

USING AN ARRAY IN SAS TO DETECT MISSING VALUES

Question:
How do I exclude observations from my PROC FREQ analysis when a value is missing from a list of variables?

Answer:
In the SAS DATA step you can create a new variable ("miss" in the example below), that is set equal to 1 when a variable has a missing value, 0 otherwise. Use the **ARRAY** statement and a **DO** loop to check for missing values across a list of variables, syntax is:

```
DATA one ;
   INFILE xxx;
   INPUT a b c d e;
   miss=0;
   ARRAY vv(5) a b c d e ;
   DO i=1 TO 5 ;
     IF vv(i)=. THEN DO;
        miss=1 ;
        i=5;
     END;
   END;
RUN;

PROC FREQ;
   WHERE miss =0;
   TABLES a b c d e ;
RUN ;
```

Here the array "vv" has 5 elements (a,b,c,d,e) and the loop "i" is likewise set to 5. For each observation the loop iterates 5 times checking for missing values across the list of 5 variables. When a missing value is encountered the variable "miss" is set to 1 and the loop stopped for that observation. "Miss" was initially set to zero, and it is only changed if an observation has missing data on any of the five variables. The **PROC FREQ** then uses the **WHERE** statement to restrict processing to observations having "miss" set to zero.

For additional information, consult the SAS Language Guide Version 8 First Edition in the Language Statement section on arrays.

ONE-TO-MANY DATASET MERGING WITH SAS

Question:
I need to merge two datasets in SAS so that the variables in one dataset are included in with the other dataset. In one dataset, observations represent companies' year-end reported data. In the other dataset, observations represent monthly activity for the year for each company in the first dataset. Thus, each observation in the first dataset will be merged with 12 observations in the second. How can I get SAS to do this kind of merge?

Answer:
To do this you must have some common variable in each dataset that uniquely identifies each company. Then you would use PROC SORT to sort the observations in each of your two SAS datasets by this "id" variable. You would then use the **MERGE** statement to interleave the two datasets together into one SAS dataset. For example:

```
PROC SORT DATA=dsetone;
  BY idvar;
RUN;
```

```
PROC SORT DATA=dsettwo;
  BY idvar;
RUN ;

DATA new;
  MERGE dsettwo dsetone ;
  BY idvar ;
RUN ;
```

where "dsetone" is your larger SAS dataset (e.g., the SAS dataset with 12 observations for each level of the id variable which you merge by) and "dsettwo" is the smaller SAS dataset containing only one "yearly" value for each level of the id variable which you merge by. Finally, "idvar" is the id variable by which the other two SAS datasets were sorted and which the **MERGE** statement now uses to merge the former datasets together (e.g., "year").

If you have more questions about the **MERGE** statement, please read about it in the SAS Language Guide, Reference, Version 8, First Edition.

USING THE SAS KEEP & DROP STATEMENTS

Question:
I'm using multiple SAS datasets in the same program, and I only need to keep a few of all of the variables in the last part of my program. Is there an easy way to keep only the variables I need to use? Should I use KEEP or DROP statements?

Answer:
One way is to use either the **KEEP** or **DROP** statements in your DATA steps. Which statement you should use will depend on how many of the variables in your SAS dataset you wish to keep. If you only wish to keep a few of many variables, then use the **KEEP** statement. If you want to drop only a few variables, use the **DROP** statement. The syntax of the two statements is very similiar (and simple).

The syntax for the **KEEP** statement is:

```
KEEP var1 var2 varN ;
```

Here is an example of using the **KEEP** statement in a SAS program:

```
DATA one ;
INFILE CARDS ;
INPUT ssn age sex weight ;
CARDS ;
```

445768976 23 m 129
487593453 35 f 112
442345213 26 m 198
;

DATA two ;
SET one ;
KEEP age weight ;
RUN ;

Here the SAS user has read in some data into dataset ONE. Then a second DATA step creates dataset TWO. The **KEEP** statement is used so that only the variables AGE and WEIGHT are included in the dataset TWO.

Alternatively, the researcher could have accomplished the same goal by replacing the **KEEP** statement with a **DROP** statement and a new variable list, indicating which variables in the first SAS dataset should be *dropped* from the new SAS dataset.

DROP ssn sex ;

For more information about the **KEEP** and **DROP** statements, consult the SAS Language: Reference, Version 8, First Edition.

COMBINING MULTIPLE LINES INTO ONE USING SAS

Question:
I want to combine multiple lines (rows) of data into one line (a single row) of data in a SAS data set. How can I do this?

Answer:
Here is some code to combine multiple lines into one line. It uses SAS's **ARRAY** function.

```
* Begin sample program ;

DATA full; *this step just creates the full dataset;
   INPUT id $ stage $ x y @@;
CARDS;
jan z 1 1 jan x 2 2 jan d 3 3 jan e 4 4 joe a 11 1 joe b 22 2 joe r 33
3
joe z 44 1 pam b 2 1 pam r 2 2 pam i 2 1 pam s 2 2
;
RUN;

PROC PRINT DATA=full;
RUN;
```

```
DATA redone;
   ARRAY u(4) newvar1-newvar4;
      DO i = 1 TO 4;
           SET full;
           u(i) = x;
      END;
   DROP i;
RUN;

PROC PRINT DATA=redone;
RUN;

* End sample program ;

Output:

OBS ID STAGE    X Y
1 jan  z        1 1
2 jan  x        2 2
3 jan  d        3 3
4 jan  e        4 4
5 joe  a        11 1
6 joe  b        22 2
7 joe  r        33 3
8 joe  z        44 1
9 pam  b        2 1
10 pam r        2 2
11 pam i        2 1
12 pam s        2 2

OBS NEWVAR1 NEWVAR2 NEWVAR3 NEWVAR4     ID     STAGE      X    Y

1       1       2       3       4       jan      e        4    4
2       11      22      33      44      joe      z        44   1
3       2       2       2       2       pam      s        2    2
```

FORMATTING SAS DATA

Question:
How do I get sas to take the number : 0001 and to output it EXACTLY as it looks without stripping off the leftmost zeros?

Answer:
If you want SAS to be able to use this value as a number, use the **Zw.d** format, where "w" is the total width of the output field to be used including any decimal point, and "d" is the number of decimal places. This format will right justify the data in the field and pad any left side blanks with zeros.

For example:

```
DATA showform;
   INPUT x ;
   FORMAT x Z5.;
CARDS;
00001
2
000003
;
RUN ;

PROC PRINT;
RUN;
```

Since the "w" value specified was 5, the output is printed with as many leading zeros as are required to fill a five character wide output field. Thus the first value is printed exactly as it is input, the second value is printed with four new leading zeros, the third value has one less leading zero.

If you don't need this value treated as a number, input it as a character value -- in this case whatever form it has in input is stored for output. However, you will not be able to use this value in any numeric procedures or calculations.

For example:

```
DATA showform;
   INPUT x $;
CARDS;
00001
;
RUN ;

PROC PRINT;
RUN;
```

For more information, please refer to the SAS LANGUAGE manual, Reference, Version 8, First Edition. Chapter 14 covers output formats.

CREATING A SAS DATA SET FROM AN ASCII FILE

Question:
How can I create a SAS data set from an ascii (raw data or text) file?

Answer:
In addition to the SAS import wizard (File, Import), ascii data files can be read into SAS using an INFILE statement within a data step.

The following steps should enable you to create a permanent SAS data set from most ascii files:

1. Look at the ascii data file in a text editor such as WordPad to determine the data layout and note what character separates the fields (*e.g.*, blank, tab, comma, tilde).
2. Modify the following example SAS code to read in your ascii data file. Note that the ascii file may be open or closed while you read the data into SAS. The most recently **saved** version is what SAS will read.

Example 1 (Windows; fixed column data layout)

Assume the data consist of five observations of three variables saved in a text file called mydata1.txt in the c:\files\data folder. The data might look like this:

```
id001    Dept of Advertising        20
id002    Dept of Communications     25
id003    Dept of Ecology            15
id004    Dept of Geology            23
id005    Dept of Health             40
```

Submit the following SAS code to read the text file:

```
FILENAME text1 "c:\files\data\mydata1.txt";

DATA mydata1;
    INFILE text1;
    INPUT var1 $ 1-5 var2 $ 10-31 var3 36-37;
RUN;
```

This code will read a fixed column ascii data file with two character variables (var1, which is contained in columns 1 through 5, and var2, which is contained in columns 10 through 31) and one numeric variable, var3, which is contained in columns 36 through 37. A temporary SAS data set (mydata1.sd2) is created and stored in the WORK directory until the SAS session is terminated.

Example 2 (Windows; comma-delimited data layout)

Assume the data consist of five observations of three variables saved in a text file called mydata2.txt in the c:\files\data folder. The data might look like this:

```
id001,Dept of Advertising,20
id002,Dept of Communications,25
id003,Dept of Ecology,15
id004,Dept of Geology,23
id005,Dept of Health,40
```
Submit the following SAS code to read the text file:

```
FILENAME text2 "c:\files\data\mydata2.txt";

DATA mydata2;
     length var1 $5 var2 $30;
     INFILE text2 dlm=',';
     INPUT var1 var2 var3;
RUN;
```

This code will read an ascii data file with two character variables (var1 and var2, which have maximum lengths of 5 and 30, respectively) and one numeric variable, var3. The variables are separated by commas in the ascii file. A temporary SAS data set (mydata2.sd2) is created and stored in the WORK directory until the SAS session is terminated.

Guidelines for reading hierarchical data are located in SAS FAQ #34.

For more complicated file layouts, refer to the INFILE options described below.

Infile Options

The INFILE statement supports various options that allow you to read ascii files with greater flexibility. Some helpful options are described below:

dlm=',' -- indicates that commas are used to separate variables within the text file. Another common delimiter is the tab. It is denoted by **dlm='09'x**. The default delimiter is a blank space.

dsd -- allows a delimiter, such as a comma, to be read as a character (instead of a delimiter) within quoted strings; also allows two consecutive delimiters to indicate a missing value. This is an appropriate option for data generated by most database software programs.

missover -- prevents SAS from going to a new input line if it does not find values in the curent line for all variables. This is an appropriate option when the data in the ascii file consist of one record per line.

notab -- allows embedded blanks within a character string.

firstobs=n -- indicates which line in the ascii file contains the first record to be read by SAS. If the first record(s) contains header information such as variable names, then set **firstobs=n** where **n** is the record number where the data actually begin.

obs=n -- indicates which line in the ascii file contains the last record to be read by SAS. This is a good option to use for testing your program. Once your program is running correctly with the first **n** observations, you can increase the value of **obs** to continue testing your program or remove the **obs=n** option altogether and read all of the records from the ascii file.

A typical INFILE statement for reading tab-delimited data located in rows 2 through 20 of a text file would be:

INFILE text1 dlm='09'x notab dsd missover firstobs=2 obs=20;

For more information on the INFILE statement, see SAS Online Help by typing INFILE or INFILE_1 under the Find tab (V6.12 under Windows).

FILE RELATIONSHIPS IN SAS

Question:
What are the different files used and produced by SAS and what are they all for?

Answer:
When running SAS in noninteractive mode, (that is when you issue the SAS command followed by an input file name), there are five different files that you are likely to be using.

These are:

1. text data files
2. command files
3. SAS data set files
4. SAS listing files
5. SAS log files

A text data file is a document that contains your data in text form. You entered the data from some text editor into this file. Note that it is possible to enter data in a command file, so separate data files are not necessary. However, since data can be read in many different ways, it is usually more efficient to create data files separate from command files.

A command file is a document that contains the SAS commands that will read the data in the text data file, plus commands that will produce some sort of output from the data. You must create the command file using a text editor program, just as with your text data file. As mentioned, the command file can also contain the data.

A SAS data set file is created when a command file contains a DATA step. SAS data set files can be created so that they exist only as long as SAS is running your current noninteractive submission, or they can be created to be saved and used again during a different submission. Note that SAS data set files cannot be edited by text editor programs.

The only way to see the data in a SAS data set file is to use the file in SAS, (or some other program that can read SAS data set files).

SAS log files are produced as SAS runs the command file. When the command file is read, SAS checks each statement and tries to execute it. If it can, the statement is sent to the SAS log file with either no comment attached, or a brief note describing the execution process. If the statement can be executed, but problems occurred during execution, the statement is sent to the SAS log file with a warning statement describing the problem. If the statement cannot be executed, SAS sends the statement to the log file along with an error message describing why the command could not be used. Examination of log files following submission of command files is the best way to find mistakes in a comand file.

Listing files are produced when SAS executes a command from the command file that creates output. Any command that creates output sends that output to the listing file. The listing file is the file with the information you were trying to produce with the command file.

There are other files used and produced by SAS. For more information, see SAS Language: Reference, Version 8, First Edition. In particular, see Chapter 2: The DATA step, and Chapter 5: SAS Output.

BIBLIOGRAPHY OF USEFUL WEB LINKS

SAS Language: Reference, Version 8, First Edition
http://v8doc.sas.com/sod_register.html

University of Texas Support for Statistical Packages (including SAS)
http://ssc.utexas.edu/consulting/answers/faqs.html

Frequently Asked Questions about SAS Software
http://www.hollandnumerics.co.uk/sasfaq/SASFAQ.HTM

SAS Tips
http://members.tripod.com/~schick/sas_tips.html