SAS 2. Getting Your Data into SAS

1. Methods for Getting Your Data into SAS

Data come in many different forms. Wherever your data reside, there is a way for SAS to use them. Methods for getting your data into SAS can be put into four general categories:

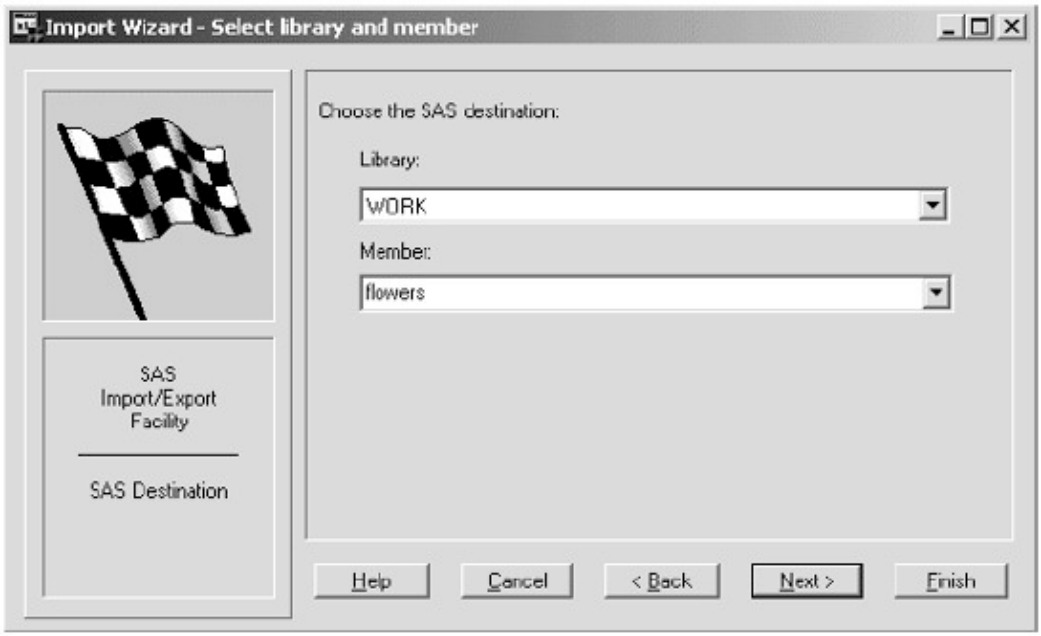
- Entering data directly into SAS data sets
- Creating SAS data sets from raw data files
- Converting other software's data files into SAS data sets
- Reading other software's data files directly.

Using the Import Wizard1, you can read a variety of data file types into SAS by simply answering a few questions. The Import Wizard will scan your file to determine variable types2 and will, by default, use the first row of data for the variable names.

2. Reading Files with the Import Wizard

We now use xlsx data to show how to

- Start the Import Wizard by choosing Import Data… from the File menu.
- Select the type of file you are importing by choosing from the list of standard data sources such as Microsoft Excel Workbook (*.xlsx) files.
- Specify the location of the file that you want to import.
- The next screen asks you to choose the SAS library and member name for the SAS data set that will be created. If you choose the WORK library, then the SAS data set will be deleted when you exit SAS. If you choose a different library, then the SAS data set will remain even after you exit SAS.



- In the last window, the Import Wizard gives you the option of saving the PROC IMPORT statements used for importing the file.

Using imported data in a SAS program Data that you import through the Import Wizard can be used in any SAS program. For example, if you saved your data in the WORK library and named it FLOWERS, you could print it with this program:

```
PROC PRINT DATA=WORK.flowers;
RUN;
```

Or, since WORK is the default library, you could also use:

```
PROC PRINT DATA=flowers;
RUN;
```

3. Telling SAS Where to Find Your Raw Data

If your data are in raw data files (also referred to as text, ASCII, sequential, or flat files), using the DATA step to read the data gives you the most flexibility. The first step toward reading raw data files is telling SAS where to find the raw data. Your raw data may be either internal to your SAS program, or in a separate file.

Internal raw data If you type raw data directly in your SAS program, then the data are internal to your program.

- Use the DATALINES statement to indicate internal data. The DATALINES statement must be the last statement in the DATA step.
- All lines in the SAS program following the DATALINES statement are considered data until SAS encounters a semicolon.
- The $ after variable names indicates that it is a character variable.

```
* Read internal data into SAS data set uspresidents;
DATA uspresidents;
INPUT President $ Party $ Number;
DATALINES;
Adams F 2
Lincoln R 16
Grant R 18
Kennedy D 35
;
RUN;
```

External raw data files Usually you will want to keep data in external files, separating the data from the program.

- Use the INFILE statement to tell SAS the filename and path, if appropriate, of the external file containing the data.
- The INFILE statement follows the DATA statement and must precede the INPUT statement.

- After the INFILE keyword, the file path and name are enclosed in quotation marks.

The following program shows the use of the INFILE statement to read the external data file:

```
* Read data from external file into SAS data set;
DATA uspresidents;
INFILE 'D:\Users\mshu\Desktop\2.txt';
INPUT President $ Party $ Number;
RUN;

PROC PRINT data=uspresidents;
RUN;
```

4. Reading Raw Data Separated by Spaces

If the values in your raw data file are all separated by at least one space,1 then using list input (also called free formatted input) to read the data may be appropriate.

- You must read all the data in a record—no skipping over unwanted values.
- Any missing data must be indicated with a period.
- Character data, if present, must be simple: no embedded spaces, and no values greater than eight characters in length.
- If the data file contains dates or other values which need special treatment, then list input may not be appropriate.

```
DATA toads;
INPUT ToadName $ Weight Jump1 Jump2 Jump3;
DATALINES;
Lucky 2.3 1.9 . 3.0
Spot 4.6 2.5 3.1 .5
Tubs 7.1 . . 3.8
Hop 4.5 3.2 1.9 2.6
Noisy 3.8 1.3 1.8
1.5
Winner 5.7 . . .
;

* Print the data to make sure the file was read correctly;
PROC PRINT DATA = toads;
TITLE 'SAS Data Set Toads';
RUN;
```

Exercise 1: Save the data in the file 'ToadJump.dat', and use the following script to read the data.

```
DATA toads;
INFILE '…(path)…\ToadJump.dat';
INPUT ToadName $ Weight Jump1 Jump2 Jump3;
PROC PRINT DATA = toads;
TITLE 'SAS Data Set Toads';
RUN;
```

5. Reading Raw Data Arranged in Columns

Some raw data files do not have spaces (or other delimiters) between all the values or periods for missing data—so the files can't be read using list input. But if each of the variable's values is always found in the same place in the data line, then you can use column input as long as all the values are character or standard numeric. Standard numeric data contain only numerals, decimal points, plus and minus signs, and E for scientific notation. Numbers with embedded commas or dates, for example, are not standard.

Column input has the following advantages over list input:

- spaces are not required between values;
- missing values can be left blank;
- character data can have embedded spaces;
- you can skip un wanted variables.

With column input, the INPUT statement takes the following form:

- After the INPUT keyword, list the first variable's name.
- If the variable is character, leave a space; then place a $.
- After the $, or variable name if it is numeric, leave a space; then list the column or range of columns for that variable. The columns are positions of the characters or numbers in the data line and are not to be confused with columns like those you see in a spreadsheet.
- Repeat this for all the variables you want to read.

```
DATA sales;
INPUT VisitingTeam $ 1-20 ConcessionSales 21-24 BleacherSales 25-28 OurHits 29-31
TheirHits 32-34 OurRuns 35-37 TheirRuns 38-
40; DATALINES;
Columbia Peaches      35  67  1 10  2  1
Plains Peanuts       210      2  5  0  2
Gilroy Garlics        151035 12 11  7  6
Sacramento Tomatoes  124  85 15  4  9  1
;

PROC PRINT DATA = sales;
TITLE 'SAS Data Set
Sales'; RUN;
```

```
                          SAS Data Set Sales                          1

                          Concession Bleacher  Our Their  Our Their
   Obs    VisitingTeam        Sales     Sales  Hits  Hits Runs  Runs

    1  Columbia Peaches         35        67     1    10    2    1
    2  Plains Peanuts          210         .     2     5    0    2
    3  Gilroy Garlics           15      1035    12    11    7    6
    4  Sacramento Tomatoes     124        85    15     4    9    1
```

Exercise 2: Save the data in the file 'Onions.dat', and use the following script to read the data.

```
DATA sales;
INFILE 'c: …(path)…\Onions.dat';
INPUT VisitingTeam $ 1-20 ConcessionSales 21-24 BleacherSales 25-28
OurHits 29-31 TheirHits 32-34 OurRuns 35-37 TheirRuns 38-40;
PROC PRINT DATA = sales; TITLE 'SAS Data Set Sales'; RUN;
```

Exercise 3: Add or delete a blank column at the 19th column, and see what happens.

6. Reading Raw Data Not in Standard Format

Sometimes raw data are not straightforward numeric or character. For example, we humans easily read the number 1,000,001 as one million and one, but computers see it as a character string. In SAS, informats are used to tell the computer how to interpret these types of data.

Informats are useful anytime you have non-standard data. (Standard numeric data contain only numerals, decimal points, minus signs, and E for scientific notation.) There are three general types of informats: character, numeric, and date. The three types of informats have the following general forms:

Character	**Numeric**	**Date**
$informatw.	informatw.d	informatw.

The $ indicates character informats, INFORMAT is the name of the informat, w is the total width, and d is the number of decimal places (numeric informats only). The period is very important part of the informat name. Use informats by placing the informat after the variable name in the INPUT statement; this is called formatted input.

Example: In the following, the variable Name has an informat of $16., meaning that it is a character variable 16 columns wide. Variable Age has an informat of three, is numeric, three columns wide, and has no decimal places. The +1 skips over one column. Variable Type is character, and it is one column wide. Variable Date has an informat MMDDYY10. and reads dates in the form 10-31-2003 or 10/31/2003, each 10 columns wide. The remaining variables, Score1 through Score5, all require the same informat, 4.1.

```
DATA contest;
INPUT Name $16. Age 3. +1 Type $1. +1 Date MMDDYY10.
(Score1 Score2 Score3 Score4 Score5) (4.1);
DATALINES;
Alicia Grossman  13 c 10-28-2003 7.8 6.5 7.2 8.0 7.9
Matthew Lee       9 D 10-30-2003 6.5 5.9 6.8 6.0 8.1
Elizabeth Garcia 10 C 10-29-2003 8.9 7.9 8.5 9.0 8.8
Lori Newcombe     6 D 10-30-2003 6.7 5.6 4.9 5.2 6.1
Jose Martinez     7 d 10-31-2003 8.9 9.510.0 9.7 9.0
Brian Williams   11 C 10-29-2003 7.8 8.4 8.5 7.9 8.0
;
PROC PRINT DATA = contest;
TITLE 'Pumpkin Carving Contest';
RUN;
```

```
DATA contest;
INPUT Name $16. Age 3. +1 Type $1. +1 Date MMDDYY10.
(Score1 Score2 Score3 Score4 Score5) (4.1);
FORMAT Date MMDDYY10.;
DATALINES;

FORMAT Date DATE9.;
```

Exercise 4: Add or delete a blank column right before 13, and see what happens.

7. Mixing Input Styles

Each of the three major input styles has its own advantages. Sometimes you use one style, sometimes another, and sometimes the easiest way is to use a combination of styles. SAS is so flexible that you can mix and match any of the input styles for your own convenience.

- When SAS reads a line of raw data it uses a pointer to mark its place, but each style of input uses the pointer a little differently.
- With list style input, SAS automatically scans to the next non-blank field and starts reading.
- With column style input, SAS starts reading in the exact column you specify.
- But with formatted input, SAS just starts reading—wherever the pointer is, that is where SAS reads.
- Sometimes you need to move the pointer explicitly, and you can do that by using the column pointer, @n, where n is the number of the column SAS should move to.

Example The following raw data contain information about U.S. national parks: name, state (or states as the case may be), year established, and size in acres:

```
DATA nationalparks;
INPUT ParkName $ 1-22 State $ Year @40 Acreage COMMA9.;
DATALINES;
Yellowstone           ID/MT/WY 1872    4,065,493
Everglades            FL 1934          1,398,800
Yosemite              CA 1864            760,917
Great Smoky Mountains NC/TN 1926         520,269
Wolf Trap Farm        VA 1966                130
;
PROC PRINT DATA = nationalparks;
TITLE 'Selected National Parks';
RUN;
```

Exercise 5: Delete and add one blank column right before 4,065,493, see what happens.

Exercise 6: In the preceding program, the column pointer @40 tells SAS to move to column 40 before reading the value for Acreage. Try the following code to see why we need @ in the INPUT.

```
DATA nationalparks;
*INPUT ParkName $ 1-22 State $ Year @40 Acreage COMMA9.;
INPUT ParkName $ 1-22 State $ Year Acreage COMMA9.;
DATALINES;
```

```
Yellowstone           ID/MT/WY 1872    4,065,493
Everglades            FL 1934          1,398,800
Yosemite              CA 1864            760,917
Great Smoky Mountains NC/TN 1926         520,269
Wolf Trap Farm        VA 1966                130
;
PROC PRINT DATA = nationalparks;
TITLE 'Selected National Parks';
RUN;
```

8. Reading Multiple Lines of Raw Data per Observation

If your data file has multiple lines of raw data per observation, it is better for you to explicitly tell SAS when to go to the next line than to make SAS figure it out. To tell SAS when to skip to a new line, you simply add line pointers to your INPUT statement.

- The line pointers, slash (/) and pound-n (#n), are like road signs telling SAS, "Go this way."
- To read more than one line of raw data for a single observation, you simply insert a slash into your INPUT statement when you want to skip to the next line of raw data.
- The #n line pointer performs the same action except that you specify the line number. The n in #n stands for the number of the line of raw data for that observation; so #2 means to go to the second line for that observation, and #4 means go to the fourth line.

Example: Read the data that contain information about temperatures for the month of July for Alaska, Florida, and North Carolina.

```
DATA highlow;
INPUT City $ State $ / NormalHigh NormalLow #3 RecordHigh RecordLow;
DATALINES;
Nome AK
55 44
88 29
Miami FL
90 75
97 65
Raleigh NC
88 68
105 50
;
PROC PRINT DATA = highlow;
TITLE 'High and Low Temperatures for July';
RUN;
```

9. Reading Multiple Observations per Line of Raw Data

When you have multiple observations per line of raw data, you can use double trailing at signs (@@) at the end of your INPUT statement. This linehold specifier is like a stop sign telling SAS, "Stop, hold that line of raw data." SAS will hold that line of data, continuing to read observations

until it either runs out of data or reaches an INPUT statement that does not end with a double trailing @.

Example: Read the data that contain the name of each city, the state, average rainfall for the month of July, and average number of days with measurable precipitation in July.

```
DATA rainfall;
INPUT City $ State $ NormalRain MeanDaysRain @@;
DATALINES;
Nome AK 2.5 15 Miami FL 6.75
18 Raleigh NC . 12
;
PROC PRINT DATA = rainfall;
TITLE 'Normal Total Precipitation and';
TITLE2 'Mean Days with Precipitation for July';
RUN;
```

10. Reading Limited FILES with the DATA Step

Delimited files are raw data files that have a special character separating data values. Many programs can save data as delimited files, often with commas or tab characters for delimiters. SAS gives you two options for the INFILE statement that make it easy to read delimited data files: the DLM= option.

Example: Save the following in the file "Books.dat" in your local machine.

```
Grace,3,1,5,2,6
Martin,1,2,4,1,3
Scott,9,10,4,8,6
```

Then use the following code to read the file.

```
DATA reading;
INFILE 'D:\Users\mshu\Desktop\Books.txt' DLM = ',';
INPUT Name $ Week1 Week2 Week3 Week4 Week5;
RUN;

PROC PRINT DATA = reading;
TITLE 'Reading Record';
RUN;
```

If the same data had tab characters between values instead of commas, then you could use the DLM='09'X option.

11. Use IMPORT Procedure to Read Files

You can use the IMPORT procedure to import several types of PC files, which includes Microsoft Excel (*.xls), Comma Separated Values (*.csv), Tab delimited file (*.txt), etc.

```
PROC IMPORT OUT= WORK.test
            DATAFILE= "\\mysbfiles.campus.stonybrook.edu\mshu\My SAS Fil
es\9.4\test.xlsx"
            DBMS=EXCEL REPLACE;
     RANGE="Sheet1$";
     GETNAMES=YES;
     MIXED=NO;
     SCANTEXT=YES;
     USEDATE=YES;
     SCANTIME=YES;
RUN;

proc print data=test;
 run;
```

```
PROC IMPORT OUT= WORK.test
            DATAFILE= "\\mysbfiles.campus.stonybrook.edu\mshu\My SAS
Files\9.4\test.xlsx"
            DBMS=EXCEL REPLACE;

RUN;
proc print data=test;
title "1";
 run;
```

```
PROC IMPORT OUT= WORK.test
            DATAFILE= " D:\Users\mshu\Desktop\test.xlsx"
            DBMS=EXCEL REPLACE;
RUN;
proc print data=test;
title "2";
 run;
```