
# READING EXTERNAL FILES INTO SAS USING A DATA STEP

## Objectives

We will study in this presentation

- How to read standard,
- non standard and
- instream data

## Introduction

As already mentioned, SAS programming alternates between the DATA step and the PROC step.

The data step reads and manipulates data which can either be in the form of a SAS dataset or an external data file

In this presentation, we will look at how SAS reads external files. In the next class we will discuss how to read SAS datasets into SAS

An external file is a file that is not a SAS datafile. It can either be a text file, an ascii, excel, access, stata file, and so on

Check CANVERS for all the dataset for this lesson. It is a good idea to save these data in a folder in your computer.

To read an external file to SAS using a DATA step, you first have to create a file name statement e.g.

```
FILENAME MODULE2 'F:\Comp Apps\Week 2\Data';
```

The filename statement has a FILEREF (which is short form for file reference), named MODULE2 that references ALL the external files.

If you click on *File Shortcuts* in the explorer window, you see the FILEREF in the *File Shortcuts*. All the external files are stored in there.


This example program reads the external file named topten that is referenced by the FILEREF module2

```
data cityrank;
  infile MODULE2('topten.txt');
  input Rank City & $12. Pop86 : comma.;
run;
```

The first 2 statements (the data and the infile statements) are mostly similar irrespective of the external file you want to read.

The third statement (which is the INPUT statement) depends on the nature of the external file.

SAS provides three primary input styles: column, formatted, and list input.

To understand why different input styles are needed, look at these different external files

1. This external file contains data that is arranged in columns or fixed fields. You can see that each field has a beginning and ending column.

| 1---+----10---+----20 |
|---|
| 2810 **61** MOD **F** |
| 2804 **38** HIGH **F** |
| 2807 **42** LOW **M** |
| 2816 **26** HIGH **M** |
| 2833 **32** MOD **F** |
| 2823 **29** HIGH **M** |

2. Just like the previous file, this external file contains data that is arranged in columns or fixed fields. However, one of the variable's values includes a special character, which is the comma (,).

Raw Data File Empdata

| 1---+----10---+----20---+--- |
|---|
| EVANS DONNY 112 29,996.63 |
| HELMS LISA 105 18,567.23 |
| HIGGINS JOHN 111 25,309.00 |
| LARSON AMY 113 32,696.78 |
| MOORE MARY 112 28,945.89 |
| POWELL JASON 103 35,099.50 |
| RILEY JUDY 111 25,309.00 |

| |
|---|
| RYAN NEAL 112 28,180.00 |
| WILSON HENRY 113 31,875.46 |
| WOODS CHIP 105 17,098.71 |

The presence of this special character causes the values for this variable to be non-standard

Nonstandard numeric data includes data with values that contain special characters, such as percent signs (%), dollar signs ($), and commas (,) ; date and time values data in fraction.

Nonstandard data values require an input style that has more flexibility than column input.

You can use **formatted input**, which combines the features of column input with the ability to read both standard and nonstandard data.

3. This external file contains data that is free-format, meaning that it is not arranged in columns. Notice that the values for a particular field do not begin and end in the same columns.

| 1---+----10---+----20 |
|---|
| MALE 27 1 8 0 0 |
| FEMALE 29 3 14 5 10 |
| FEMALE 34 2 10 3 3 |
| MALE 35 2 12 4 8 |
| FEMALE 36 4 16 3 7 |
| MALE 21 1 5 0 0 |
| MALE 25 2 9 2 1 |
| FEMALE 21 1 4 2 6 |
| MALE 38 3 11 4 3 |
| FEMALE 30 3 5 1 0 |

COLUMN INPUT → used in conjunction with standard data

Let's look at how column input can be used to read the previous data just shown.

```
data excercise;
  infile MODULE2('exer.txt');
  input ID $ 1-4 Age 6-7 ActLevel $ 9-12 Sex $ 14;
run;
```

The data step starts with the keyword DATA followed by the name of the dataset (EXERCISE). The infile statement references the FILEREF MODULE2 created in the FILENAME statement.

The INPUT statement assigns the variable ID to the data in columns 1-4 (since ID is a character variable, a dollar sign '$' is placed after the name of the variable). The numeric variable Age is written to the data in columns 6-7. The character variable, ActLevel occupies columns 9-12 and and the character variable Sex occupies column 14.

To view this dataset, use this proc print step

```
Proc print data=exercise;
Run;
```

FORMATED INPUT → used in conjunction with non-standard data

The EmpData introduced at the beginning of this lesson contains personnel information for the technical writing department of a small computer manufacturer. See that the data is arranged in column,that is you can determine a start and end column for each variable.

The fields contain values for each employee's last name, first name, job title, and annual salary.

The COLUMN input cannot read this data because as already explained, the data values of the variable, (annual salary) are nonstandard (data that contains values with special characters, such as percent signs (%), dollar signs ($), and commas (,) ; date and time values data in fraction).

Nonstandard data values require an input style that has more flexibility than column input.

You can use **formatted input**, which combines the features of column input with the ability to read both standard and nonstandard data.

Formatted input uses the **column pointer controls to move the input pointer to a particular column**. There are 2 input pointer controls:

1. @n pointer control→ move a pointer forward or backward when reading a record.
2. +n pointer control

This program uses the @n input pointer control to read the external file EmpData.


```
data EmpData;
  infile MODULE2('EmpData.txt');
  input LastName $7. @9 FirstName $5. @15 JobTitle 3. @19 Salary comma9.;
run;
```

Alternatively, a similar program with the +n pointer control is as follows:

```
data EmpData;
  infile MODULE2('EmpData.txt');
  input LastName $7. +1 FirstName $5. +5 Salary comma9. @15 JobTitle 3.;
run;
```

## LIST INPUT

Reading Standard Free format data

This program reads a free format dataset,

```
data survey;
  infile MODULE2('credit.txt');
  input gender $ Age Bankcard FreqBank DeptCard;
run;
```

This input statement known as the list input reads the free format dataset by simply listing the variable names: gender, age, number of bank credit cards, bank card use per month, number of department store credit cards and department store card use per month; in the same order as the corresponding raw data fields.

Because list input, by default, does not specify column locations, all fields in the external file must be separated by at least **one** blank or other delimiter, fields must be read **in order** from left to right and you cannot skip or re-read fields.

The names in the program are listed exactly as they appear in the external file. A dollar sign ($) is included when the variable is character

Display the data set with this PRINT procedure.

```
proc print data=perm.survey;
run;
```

### Reading nonstandard free format data


Let's take a look at this dataset. This file contains the names of the ten largest U.S. cities ranked in order based on their 1986 estimated population figures. See that the file contains data that it is not arranged in columns. Notice that on like the previous free format data, some of the entries for the first column in this data have embedded blanks. Also, notice the values representing the 1986 population of each city in the raw data file below. Because they contain commas, these values are nonstandard numeric values. The list input introduced earlier cannot be used to read this data

Raw Data File Topten

| 1---+----10---+----20---+-- |
|---|
| 1 NEW YORK **7,262,700** |
| 2 LOS ANGELES **3,259,340** |
| 3 CHICAGO **3,009,530** |
| 4 HOUSTON **1,728,910** |
| 5 PHILADELPHIA **1,642,900** |
| 6 DETROIT **1,086,220** |
| 7 SAN DIEGO **1,015,190** |
| 8 DALLAS **1,003,520** |
| 9 SAN ANTONIO **914,350** |
| 10 PHOENIX **894,070** |

This program [known as modified list input (& and :)] can be used to read the external file

```
data cityrank;
  infile lesson1('topten.txt');
  input Rank City & $12. Pop86 : comma.;
run;
```

The input statement reads the value for rank, city (because city has embedded blanks), the ampersand (&) modifier is used to read such values. Observe that a length of 12 is assigned to the variable and since the variable city is character, a $ is needed

To read the next column which has character values longer than 8 characters and the data values contain comma (,) which make them nonstandard, we used the colon (:) modifier followed by the informat comma (notice the period at the end of each of the informats)

**Reading a Range of Variables**


This raw data file called SURVEY has an ID and 5 test questions (that have similar variables) from a survey named question1—question5.

Raw Data File Survey

| 1---+----10---+----20 |
|---|
| 1000 23 94 56 85 99 |
| 1001 26 55 49 87 85 |
| 1002 33 99 54 82 94 |
| 1003 71 33 22 44 92 |
| 1004 88 49 29 57 83 |

Rather than reading each variable one after the other, you can specify a range as follows.

```
data cityrank;
  infile module2('survey.txt');
  input IDnum $ Ques1-Ques5;
run;
```

A range can also be specified in the VAR statement in the PROC PRINT step to select only some specific variables.

```
   proc print data=cityrank;
     var ques1-ques3;
   run;
```

Options in the INFILE statement

To enhance you program, there are some SAS options that can be included in the infile statement as needed.

Some of these options include the DSD, MISSOVER, DLM=, PAD and so on. Please, see SAS documentations for a complete list of these options

DLM= Option or **Working with Delimiters**

Most free-format data fields are clearly separated by blanks as we have seen before and are easy to imagine as variables and observations. But fields can also be separated by other characters called delimiters, such as commas, or by the # sign or any other delimiter

| 1---+----10---+----20 |
|---|
| MALE,27,1,8,0,0 |

| |
|---|
| FEMALE,29,3,14,5,10 |
| FEMALE,34,2,10,3,3 |
| MALE,35,2,12,4,8 |
| FEMALE,36,4,16,3,7 |
| MALE,21,1,5,0,0 |
| MALE,25,2,9,2,1 |
| FEMALE,21,1,4,2,6 |
| MALE,38,3,11,4,3 |
| FEMALE,30,3,5,1,0 |

Adding this option DLM= in the infile statement as in the program below will read such external files. Observe that the character variable that separates the data values is placed in quotation.

```
Data credit;
      infile module2 ('Credit1.txt') dlm=',';
      input Gender $ Age Bankcard FreqBank
           Deptcard FreqDept;
run;
   proc print data= credit;
   run;
```

**The MISSOVER Option→Reading Missing Values at the End of a Record**

| 1---+----10---+----20 |
|---|
| MALE 27 1 8 0 0 |
| FEMALE  3 14 5 10 |
| FEMALE 34 2 10 |
| MALE 35 2 12 4 8 |
| FEMALE 36 4 16 3 7 |
| MALE 21 1 5 0 0 |
| MALE 25 2 9 2 1 |
| FEMALE 21 1 4 2 6 |
| MALE 38 3 11 4 3 |
| FEMALE 30 3 5 1 0 |

When data values are missing at the end of a record as in the dataset above where the third person represented in the raw data file did not answer the questions about how many department store credit cards she has and about how often she uses them, use this program with the **MISSOVER** option in the infile statement.


```
Data survey;
      infile module2 ('Credit_miss.txt') missover;
      input Gender $ Age Bankcard FreqBank
           Deptcard FreqDept;
run;

proc print data=survey;
run;
```

**DSD option:→Reading Missing Values at the Beginning or Middle of a Record**

Remember that the MISSOVER option works only for missing values that occur at the end of the record. A different method is required when the missing values occur at the beginning or middle of a record. Let's look at what happens when a missing value occurs at the beginning or middle of a record.

Suppose the value for Age is missing in the first record.

| 1---+----10---+----20 |
|---|
| MALE,,1,8,0,0 |
| FEMALE,29,3,14,5,10 |
| FEMALE,34,2,10,3,3 |
| MALE,35,2,12,4,8 |
| FEMALE,36,4,16,3,7 |
| MALE,21,1,5,0,0 |
| MALE,25,2,9,2,1 |
| FEMALE,21,1,4,2,6 |
| MALE,38,3,11,4,3 |
| FEMALE,30,3,5,1,0 |

Run this data step in SAS

```
Data survey;
      infile module2 ('Credit_miss.txt') dsd;
      input Gender $ Age Bankcard FreqBank
           Deptcard FreqDept;
run;

proc print data=survey;
run;
```

Note that when the dsd option is used, the default delimiter is the comma. So a dlm= option is not needed when a DLM= option is used

Other options of the infile statement are

1. The **OBS**= option can be added to the infile statement to limit processing to only n records
2. **The PAD** option which adds blanks to lines that are shorter than what the input statement specifies
3.

### Reading Instream Data

So far, we have focused on the INFILE statement that identifies an **external file** to read. However, you can also read **instream data lines** that you enter directly in your SAS program, rather than data that is stored in an external file. Reading instream data is extremely helpful if you want to create data and test your programming statements on a few observations that you can specify according to your needs.

Let's look at this simple program to demonstrate how to read instream data

```
Data c;
Input Num VarA $;
Datalines;
1 A1
3 A2
5 A3
;
```

To read instream data, You require

a **DATALINES statement** that precedes the input data and

a **null statement** (a single semicolon) to indicate the end of the input data.

**Example**

Suppose we want to read a dataset with the record layout for a raw data file that contains readings from exercise stress tests that have been performed on patients at a health clinic. Exercise physiologists in the clinic use the test results to prescribe various exercise therapies. The file contains **fixed fields as** values for each variable are in the same location in all records.


| Field Name | Starting Column | Ending Column | Description of Field |
|---|---|---|---|
| ID | 1 | 4 | patient ID number |
| Name | 6 | 25 | patient name |
| RestHR | 27 | 29 | resting heart rate |
| MaxHR | 31 | 33 | maximum heart rate during test |
| RecHR | 35 | 37 | recovery heart rate after test |
| TimeMin | 39 | 40 | time, complete minutes |
| TimeSec | 42 | 43 | time, seconds |
| Tolerance | 45 | 45 | comparison of stress test tolerance between this test and the last test (I=increased, D=decreased, S=same, N=no previous test) |

```
data stress;
input ID 1-4 Name $ 6-25 RestHR 27-29 MaxHR 31-33
RecHR 35-37 TimeMin 39-40 TimeSec 42-43
Tolerance $ 45;
if tolerance='D';
datalines;
2458 Murray, W          72  185 128 12 38 D
2462 Almers, C          68  171 133 10  5 I
2501 Bonaventure, T      78  177 139 11 13 I
2523 Johnson, R         69  162 114  9 42 S
2539 LaMance, K         75  168 141 11 46 D
2544 Jones, M           79  187 136 12 26 N
2552 Reberson, P        69  158 139 15 41 D
2555 King, E           70  167 122 13 13 I
2563 Pitts, D          71  159 116 10 22 S
2568 Eberhardt, S       72  182 122 16 49 N
2571 Nunnelly, A        65  181 141 15  2 I
2572 Oberon, M          74  177 138 12 11 D
2574 Peterson, V        80  164 137 14  9 D
2575 Quigley, M         74  152 113 11 26 I
2578 Cameron, L         75  158 108 14 27 I
2579 Underwood, K       72  165 127 13 19 S
2584 Takahashi, Y       76  163 135 16  7 D
2586 Derber, B          68  176 119 17 35 N
2588 Ivan, H           70  182 126 15 41 N
2589 Wilcox, E          78  189 138 14 57 I
2595 Warren, C          77  170 136 12 10 S
;
```


## Creating SAS datasets from external files using procedure step

SAS can also create a SAS data set directly using a procedure step such as proc import.

For example this **proc import** program can be used to read an excel spread sheet

```
proc import dbms=excel datafile='F:\Comp Apps\Week 2\Files\Credit.xls' out=xlsdata;
run;
```

From the above program, the dbms= EXCEL instructs SAS to read an excel spread sheet.

The datafile option directs SAS to the folder where the data is saved

The SAS dataset that is a result of this program is called xlsdata and it is obtained using this OUT option.

To convert the SPSS datafile to a SAS dataset, use this prgram

```
proc import dbms=sav   datafile='F:\Comp Apps\Week 2\Files\Credit.sav' out=spssdata;
run;
```

Here is a list of DBMS=options that can be used for these different external files

| Options of DBMS | External files | File type |
|---|---|---|
| EXCEL | Microsoft Excel | .xls |
| ACCESS | Microsoft Access | .mdb |
| CSV | Delimited file with comma-separated values | .csv |
| SAV | SPSS file | .sav |
| DLM | Delimited file (default delimiter is a blank) | .dta |
| DTA | Stata file | .dta |
| TAB | Delimited file (tab-delimited values) | .txt |

## Importing external files using the import wizard

As an alternative to using programming statements, you can use the Import Wizard to guide you through the process of creating a SAS data set from raw data. The Import Wizard is a point-and-click interface that enables you to create a SAS data set from different types of external files, such as

- dBase files (*.dbf)
- Excel 97 or 2000 Spreadsheets (*.xls)

- Microsoft Access tables
- Delimited files (*.*)
- Comma-separated values (*.csv).

To convert the excel spreadsheet to a SAS dataset using the Import Wizard, select **File** from the menu bar ▸ **Import Data (verify that 'Standard data source' is checked)** ▸ **from the drop down menu, select a data file that you want to input→in this case, "select Microsoft Excel Workbook ('xls' and so on ) →click next→browse to the folder were the data is saved and select 'Open' →click OK→click NEXT→in the member box, enter then name of the SAS dataset that you want…for example, I will enter CreditIMP.**

If you need additional information, select the **Help** button at the bottom of each screen in the wizard.

Other modifiers (+, #, @, @@);

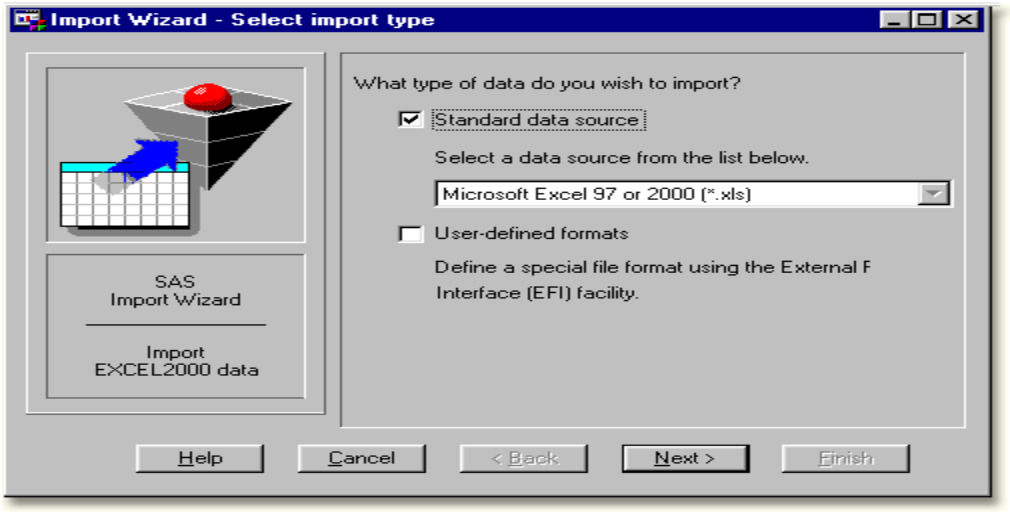
## Importing Data

As an alternative to using programming statements, you can use the Import Wizard to guide you through the process of creating a SAS data set from raw data. The Import Wizard is a point-and-click interface that enables you to create a SAS data set from different types of external files, such as

- dBase files (*.dbf)
- Excel 97 or 2000 Spreadsheets (*.xls)
- Microsoft Access tables
- Delimited files (*.*)
- Comma-separated values (*.csv).

The data sources that are available to you depend on which SAS/ACCESS products you have licensed. If you do not have any SAS/ACCESS products licensed, the only type of data source files available to you are CSV files, TXT files, and delimited files.

To access the Import Wizard, select **File** ▸ **Import Data** from the menu bar. The Import Wizard opens with the **Select import type** screen.



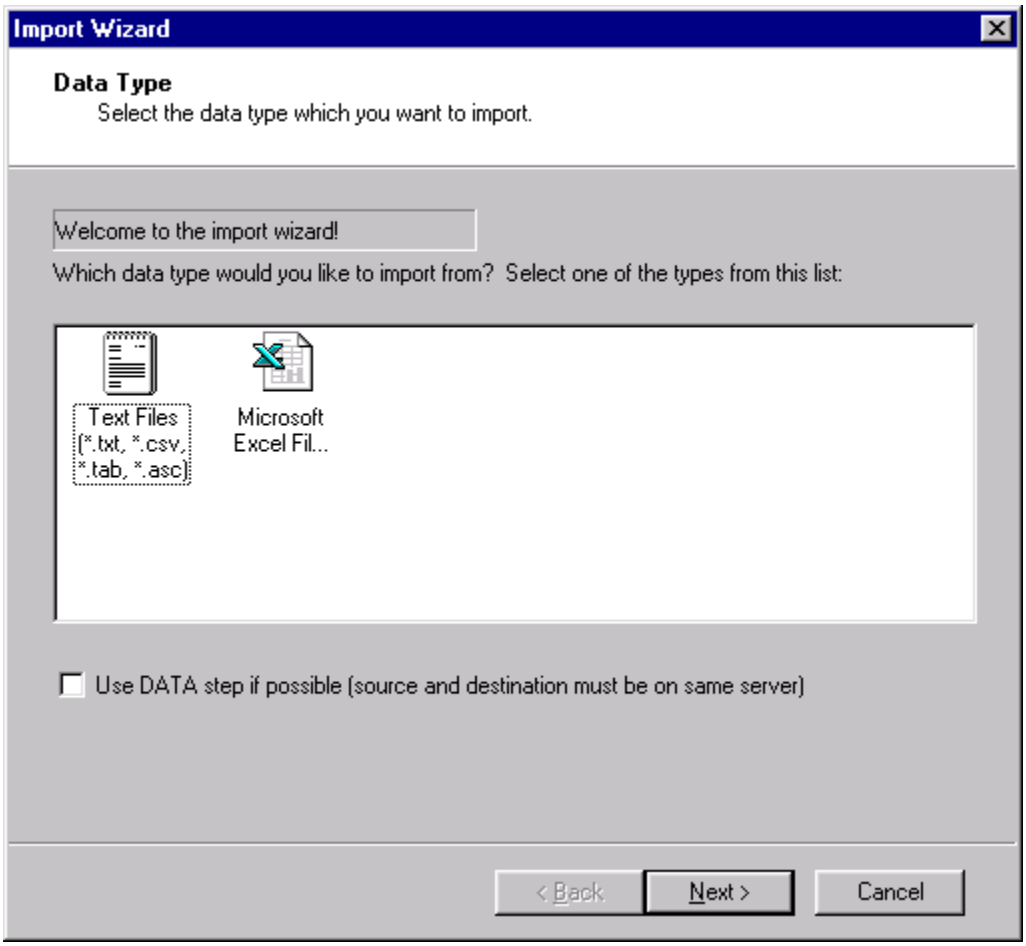

Follow the instructions on each screen of the Import Wizard to read in your data. If you need additional information, select the **Help** button at the bottom of each screen in the wizard.

Just as you can create a SAS data set from raw data by using the Import Wizard, you can use the Export Wizard to read data from a SAS data set and to write the data to an external data source. To access the Export Wizard, select **File ▸ Export Data** from the menu bar.

As an alternative to using programming statements, you can use the Import Wizard to guide you through the process of creating a SAS data set from raw data. The Import Wizard is a point-and-click interface that enables you to create a SAS data set from different types of external files, such as

- Excel Spreadsheets (*.xls)
- Text files (*.txt,*.csv, *.tab, *.asc).

To access the Import Wizard, select **Tools ▸ Import Data**. The window opens to the **Data Type** page.



Follow the instructions on each screen of the Import Wizard to read in your data.

As an alternative to using programming statements, you can use the Import Data task to guide you through the process of creating a SAS data set from raw data. The Import Data task is a point-and-click interface that enables you to create a SAS data set from different types of external files, such as

- Excel Spreadsheets (*.xls)
- Text files (*.txt,*.csv, *.tab, *.asc).

To access the Import Data window, select **Tools ▸ Import Data**. The window opens to the **Data Type** page.