How to use PROC IMPORT in SAS?

Authored by

stats writer

November 19, 2025

The PROC IMPORT is a fundamental SAS procedure designed to streamline the process of loading external data sources--such as comma-separated values (CSV), spreadsheet files, or simple text files--directly into native SAS datasets. This powerful utility significantly simplifies data integration, making it possible for analysts to quickly prepare and analyze data that originates outside the SAS environment.

Utilizing PROC IMPORT is highly efficient and requires minimal setup. At its core, it only needs the file path and name of the external data source, though various optional parameters allow for precise configuration of how the data is read, including specifying delimiters, handling variable names, and managing existing files.

Fundamentals of the PROC IMPORT Statement

To successfully transfer external data into the SAS system, you must employ the **PROC IMPORT** statement. This procedure acts as the primary interface for reading diverse file formats and transforming them into readable SAS tables.

The following structure represents the foundational syntax required for nearly all data import operations:

```
proc import out=my_data
datafile="/home/u13181/my_data.csv"
dbms=csv
replace;
getnames=YES;
run;
```

Each statement within this block serves a specific and crucial role in defining the source, destination, and characteristics of the imported data:

out: Designates the name that the resulting SAS dataset will be assigned once the import process is complete. This name must adhere to standard SAS naming conventions.
datafile: Specifies the exact location (path and filename) of the external file that is to be imported into the SAS environment.
dbms: This critical argument defines the file type or format of the source data being imported, informing SAS how to read and parse the content correctly.
replace: An optional argument that instructs SAS to overwrite the output dataset if a dataset with the same name already exists in the target library. If omitted, the procedure will fail if the dataset already exists.
getnames: Determines whether the first row of the external file should be treated as the variable

names (column headers) in the resulting SAS dataset. Set to **YES** to use the header row, or **NO** otherwise.

Specifying the Data Source Management System (DBMS)

The general syntax shown above is robust enough to handle the import of virtually any standard external file format into SAS. The only component that must change based on the input file is the value assigned to the **dbms** argument. The DBMS specification ensures that SAS uses the correct engine and parsing rules for the data structure being read.

Understanding the correct **dbms** value is essential for successful data integration. If the wrong specification is used, SAS may misinterpret the structure of the data, leading to incorrect variable types, missing values, or complete failure of the import process.

Common file types and their corresponding **dbms** arguments include:

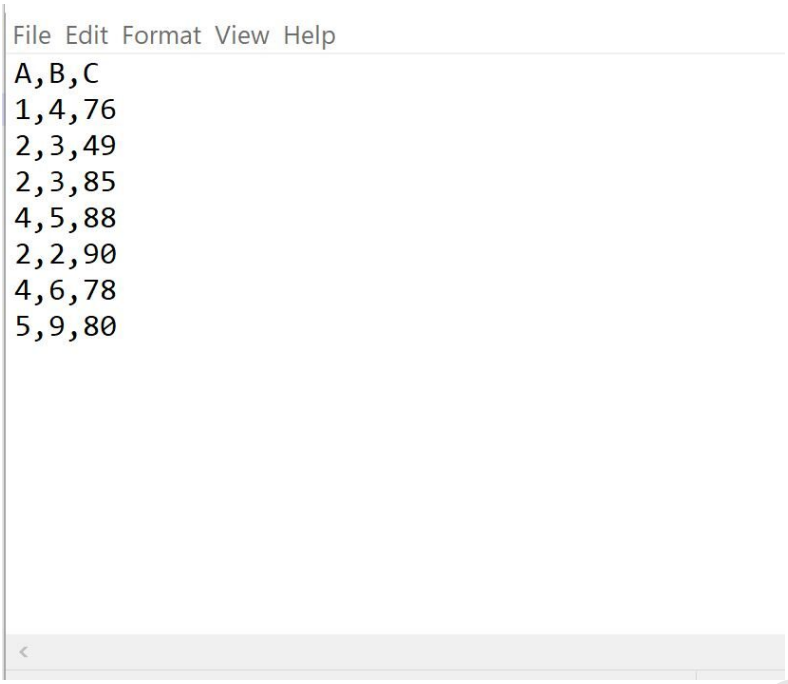
To import a standard CSV file, specify **dbms=csv**.
To import a modern Microsoft Excel workbook, specify **dbms=xlsx**.
To import a general delimited text file (using tabs, pipes, or other non-comma delimiters), specify **dbms=dlm**.

The following examples illustrate the application of PROC IMPORT across these three primary file types, demonstrating how the syntax is adjusted for each scenario.

Example 1: Using PROC IMPORT to Handle CSV Files

Consider a scenario where we need to import a file named **my_data.csv**. CSV files are inherently straightforward, as they use commas to separate values and typically organize data into a rectangular structure, making them one of the most common formats for data exchange.

Suppose the structure of our external CSV file looks like this:



We utilize the **dbms=csv** option, setting the output name to **new_data** and ensuring the first row is used for variable identification via **getnames=YES**. The **replace** option guarantees that the procedure will execute even if **new_data** already exists.

```
/*import data from CSV file called my_data.csv*/
proc import out=new_data
datafile="/home/u13181/my_data.csv"
dbms=csv
replace;
getnames=YES;
run;

/*view dataset*/
proc print data=new_data;
```

After the execution of the import and print procedures, the resulting SAS dataset **new_data** accurately reflects the content and structure of the source CSV file:

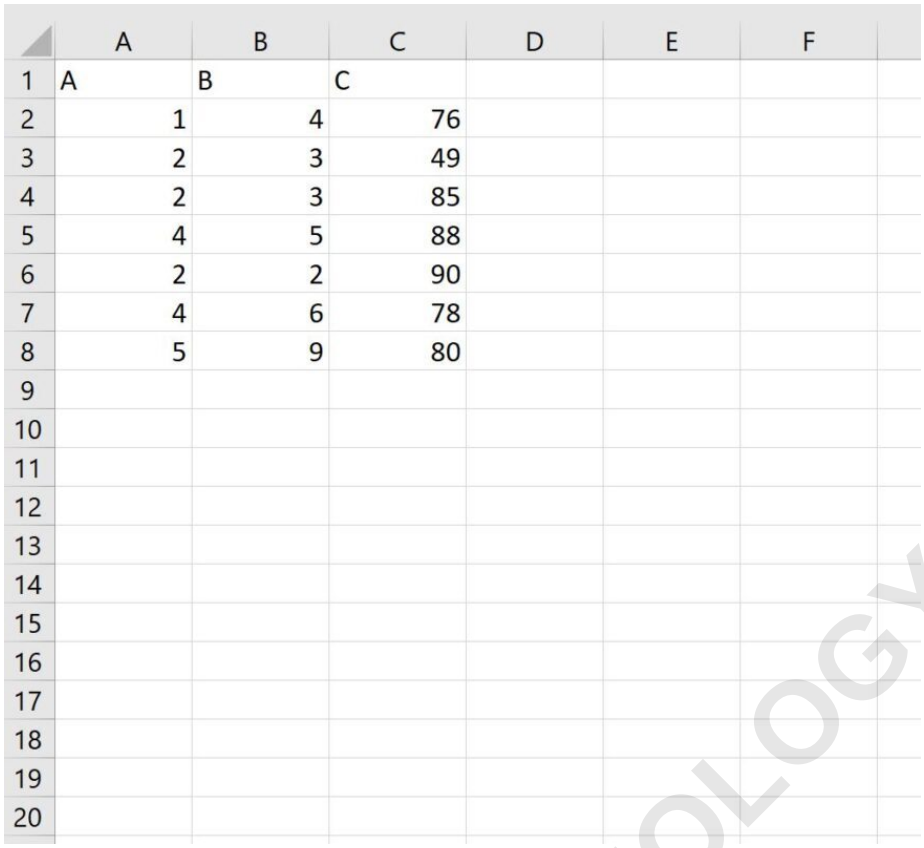
Obs	A	B	C
1	1	4	76
2	2	3	49
3	2	3	85
4	4	5	88
5	2	2	90
6	4	6	78
7	5	9	80

As shown in the output, the imported data now resides within the SAS environment, ready for subsequent analytical tasks.

Example 2: Leveraging PROC IMPORT for Excel Workbooks

Importing data from Microsoft Excel files (typically .xlsx or .xls extensions) requires a different approach than plain text files due to the complex internal structure of spreadsheets, which can include multiple sheets, specific formatting, and metadata. For this, we use **dbms=xlsx**.

Suppose we are working with an Excel file containing the following table structure:



	A	B	C	D	E	F
1	A	B	C			
2	1	4	76			
3	2	3	49			
4	2	3	85			
5	4	5	88			
6	2	2	90			
7	4	6	78			
8	5	9	80			
9						
10						
11						
12						
13						
14						
15						
16						
17						
18						
19						
20						

We utilize the following code block to import this data into SAS, again naming the output dataset **new_data**. Note that the **dbms** argument is specifically set to **xlsx** to correctly invoke the appropriate SAS engine for reading proprietary Excel formats.

```
/*import data from Excel file called my_data.xlsx*/
proc import out=new_data
datafile="/home/u13181/my_data.xlsx"
dbms=xlsx
replace;
getnames=YES;
run;

/*view dataset*/
proc print data=new_data;
```

Upon successful execution, the procedure outputs the following SAS dataset, confirming that the structure and content of the Excel sheet have been accurately transferred:

Obs	A	B	C
1	1	4	76
2	2	3	49
3	2	3	85
4	4	5	88
5	2	2	90
6	4	6	78
7	5	9	80

It is important to remember that when dealing with Excel files, additional optional parameters (such as the **SHEET=** argument) may be necessary if the desired data table is not located on the first worksheet of the workbook.

Example 3: Importing Delimited Text Files using DBMS=DLM

While CSV files are delimited by commas, many text files use alternative separators, such as tabs, pipes (|), or semicolons. These general delimited files require the **dbms=dlm** specification, where DLM stands for Delimiter. When using **dbms=dlm**, SAS can automatically detect common delimiters or be explicitly told which delimiter to use via the optional **DELIMITER=** statement (though it is omitted in this basic example).

Suppose we have the following text file, **data.txt**, which might be separated by spaces or tabs:

```
column1 column2
1 4
3 4
2 5
7 9
9 1
6 3
4 4
5 2
4 8
6 8
```

To correctly import this dataset into SAS, we utilize **dbms=dlm**, informing the PROC IMPORT procedure that it must determine the appropriate text separator:

```
/*import data from text file called data.txt*/
proc import out=new_data
datafile="/home/u13181/data.txt"
dbms=dlm
replace;
getnames=YES;
run;

/*view dataset*/
proc print data=new_data;
```

The resulting SAS output confirms the successful parsing of the delimited text file into structured variables:

Obs	column1	column2
1	1	4
2	3	4
3	2	5
4	7	9
5	9	1
6	6	3
7	4	4
8	5	2
9	4	8
10	6	8

The data displayed in the SAS output perfectly aligns with the content organized within the original text file. This confirms the flexibility of **PROC IMPORT** in handling various types of flat files.

Further Reading and Advanced Options

While these basic examples cover the most frequent uses of PROC IMPORT, the procedure offers extensive functionality to handle complex data scenarios. Arguments like **DATAROW=** (to skip initial rows of metadata), **GUESSINGROWS=** (to increase the sample size for determining variable types), and **DELIMITER=** (for specific delimited files) provide granular control over the import process.

Note: Refer to the official SAS documentation for a complete and authoritative list of optional arguments you can use when importing files, as specific requirements may vary based on the SAS version and operating environment.

Related SAS Data Management Tutorials

The following tutorials explain how to perform other common tasks in SAS: