How to import text files into SAS?

Authored by

stats writer

November 19, 2025

Welcome to the world of SAS, where efficient data handling is paramount for statistical analysis and reporting. Before any meaningful analysis can occur, raw data must be successfully loaded into the SAS environment. Text files--including common formats like CSV (Comma-Separated Values) or simple tab-delimited files--are the most frequent source of external data encountered by data scientists and analysts.

The process of importing a text file into SAS is foundational. Fortunately, SAS offers robust and flexible methods to accomplish this. Mastering these techniques ensures that your data preparation phase is swift and error-free, setting the stage for advanced data manipulation and statistical procedures.

Introduction to Text File Import in SAS

Importing external data, particularly flat files, is a critical initial step in any SAS project. The software provides two primary mechanisms for loading data from a simple text file into a structured SAS dataset: the automated PROC IMPORT procedure and the highly customizable DATA step. While both methods achieve the same objective--creating a permanent or temporary SAS dataset--they differ significantly in their required coding complexity and flexibility.

The choice between the two often depends on the complexity and cleanliness of the source text file. For standard, well-structured files, such as those in CSV format, the PROC IMPORT method is almost universally preferred due to its simplicity and efficiency. Conversely, if the data is irregularly formatted, requires complex transformations during import, or lacks standard delimiters, the DATA step offers the fine-grained control necessary to handle such challenges.

Regardless of the method selected, it is crucial that the external text file conforms to a supported delimited format. This includes common types such as CSV (comma-separated), tab-delimited, or space-delimited structures. Once successfully imported, the resulting SAS dataset becomes readily available for use across any subsequent SAS procedure, including reporting, statistical modeling, and further manipulation.

Understanding the Two Primary Import Methods

The most straightforward approach for importing text files is using the PROC IMPORT statement. This procedure is designed to automatically read the structure and contents of standard delimited files, inferring variable types and lengths based on the data it encounters. This automation dramatically reduces the amount of coding required, especially for large datasets with many columns. It acts as a wrapper that simplifies the underlying data reading process, making it highly accessible even for novice SAS users.

In contrast, the DATA step offers maximum control but demands explicit instruction from the user. When employing the DATA step with an **INFILE** statement, the user must manually specify key attributes for every variable being read: the **data type** (character or numeric), the desired **length**, and optional elements like **labels** and **formats**. While this requires more complex coding, it is indispensable when dealing with fixed-width files or files that contain missing values or formatting errors that confuse automated procedures.

For the purposes of importing standard delimited files, such as those commonly exported from spreadsheet applications, we focus primarily on the PROC IMPORT statement, as it represents the fastest and most efficient solution for the majority of data import tasks.

Deep Dive into the PROC IMPORT Statement

The PROC IMPORT procedure is a high-level utility designed specifically for reading external data files, including text files, Excel sheets, and database tables, into SAS datasets. When importing a text file, the procedure requires several critical parameters to execute correctly, allowing it to locate the source file, define the target dataset, and understand the format of the data structure.

To quickly import data from a text file into SAS, you must use the **PROC IMPORT** statement followed by the necessary directives. The basic syntax involves defining the output dataset name, specifying the input file path, and identifying the database management system (DBMS) that corresponds to the file format. For standard delimited text files, the DBMS is typically set to **DLM** (Delimited) or **CSV**.

This procedure uses the following basic syntax structure:

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

This generic template demonstrates the mandatory and highly recommended options for successful text file importation. Understanding each option is essential for troubleshooting and adapting the code to various file structures.

Essential Parameters for PROC IMPORT

The functionality of the PROC IMPORT statement relies on several key sub-statements that direct

SAS on how to handle the data source and destination. Here is a detailed breakdown of the parameters used in the standard syntax for delimited files:

out: This parameter specifies the name of the new SAS dataset that will be created upon successful import. This must be a valid SAS name and can optionally include a library reference (e.g., `LIBREF.dataset_name`) to store the data permanently.
datafile: This is the critical parameter that defines the absolute path and filename of the external text file you intend to import. It must be enclosed in quotation marks, and the path must be accessible by the SAS session being run.
dbms: This option specifies the type of data file being imported. When importing simple text files, you typically use `DLM` for files that rely on generic delimiters (like spaces, tabs, or semicolons), or `CSV` specifically for comma-separated files. If `dbms=DLM` is used, you often need to define the exact delimiter using the **DELIMITER=** option, although if the delimiter is a space, it is often inferred.
replace: This optional but highly recommended statement instructs SAS to overwrite the existing output dataset if one with the same name already resides in the target library. If `REPLACE` is omitted and the dataset exists, the procedure will fail to execute, preventing accidental data loss.
getnames: This is a binary option, set to either `YES` or `NO`. When set to **YES**, SAS treats the first row of the input text file as the names for the variables in the resulting SAS dataset. If set to **NO**, SAS assigns default variable names (e.g., `VAR1`, `VAR2`, etc.).

Using these parameters together allows the **PROC IMPORT** statement to efficiently process and structure the raw data, converting the string-based content of the text file into the optimized binary format of a SAS dataset.

Practical Example: Importing a Delimited Text File

To illustrate the efficiency of **PROC IMPORT**, let us walk through a concrete scenario. Suppose we have a simple text file named **data.txt** that contains student records. This file uses spaces as **delimiters** and includes a header row containing the variable names.

The structure of our hypothetical input file, **data.txt**, is detailed below:

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

Our goal is to import this raw data into a SAS dataset named **new_data**, ensuring that the column headers (Name, Grade, Score) are correctly utilized as variable names. Since this file is space-delimited, we will specify `dbms=dlm`.

We can use the following code block to execute this import and then immediately view the imported dataset using **PROC PRINT**:

```
/*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 successful execution of this code yields the following output in the SAS environment,

confirming the transformation of the raw text data into a structured table:

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

As visible in the output, the imported data in the SAS dataset mirrors the data contained within the original text file precisely. Note that the success of this import hinged on the use of **getnames=YES**, which correctly instructed SAS to interpret the first line of the input file as column headers rather than data rows.

Alternative Method: Using the DATA step for Complex Files

While PROC IMPORT handles most standard files, specialized scenarios--such as importing data that has highly irregular formatting, includes multiple header or footer rows that must be skipped, or requires explicit control over variable lengths--necessitate the use of the DATA step combined with the **INFILE** statement. This method is slower to write but guarantees full control over the reading process.

When using the DATA step, you must first define the input path using **INFILE**. Crucially, you then use the **INPUT** statement to define the variables, their order, and their informat (how SAS should read them). This allows you to specify whether a variable is character (using the $) or numeric, and exactly how many columns or characters it spans.

For example, if importing a CSV file using the DATA step, you would use options like **DLM=','** and **DSD** (which handles quoted strings and sequential delimiters), providing a robust alternative when automated procedures fall short. Although more verbose, this technique is the gold standard for handling non-standard or 'dirty' data sources.

Handling Delimiters and File Types

The effectiveness of PROC IMPORT relies heavily on correctly identifying the **delimiter**--the character that separates the data fields within the text file. Common delimiters include commas (for CSV files), tabs, spaces, or semicolons.

If your file is a standard CSV, setting **dbms=CSV** is usually sufficient, as SAS defaults the delimiter to a comma. However, if you have a file delimited by tabs, pipes (|), or semicolons, you must use **dbms=DLM** and explicitly define the delimiter character using the **DELIMITER=** option. For instance, for a tab-delimited file, you would include `DELIMITER='09'x`, where '09'x represents the hexadecimal code for the tab character.

A frequent issue in importing data is dealing with data values that contain the delimiter character itself (e.g., a city name like "St. Petersburg, FL" in a comma-delimited file). PROC IMPORT generally handles this gracefully if the fields are properly enclosed in quotation marks, but if issues arise, reverting to the DATA step with the **DSD** option offers greater resilience against such formatting ambiguities.

Conclusion and Next Steps in SAS Data Handling

The ability to reliably import external text files is the cornerstone of any analytical workflow in SAS. The PROC IMPORT procedure offers a powerful, low-code solution for standard files, relying on key parameters like `OUT`, `DATAFILE`, and `GETNAMES` to quickly convert raw data into a usable SAS dataset.

For users facing more complex data structures, remember that the DATA step remains the ultimate tool for explicit control over variable definitions and data reading logic. Choosing the right method based on the data source characteristics is vital for maintaining data integrity and procedural efficiency.

For those seeking deeper knowledge and advanced features, official documentation provides comprehensive insights into all parameters. We highly recommend consulting the comprehensive documentation for the **PROC IMPORT** statement to explore additional options such as handling character encoding and managing data type conflicts.

Related SAS Tutorials

The following tutorials explain how to perform other common tasks in SAS, building upon the foundational skill of data import:

How to export SAS datasets to other formats.

Methods for performing data merging and concatenation in SAS.
Introduction to statistical analysis using **PROC GLM** and **PROC REG**.