Learning SAS: A Step-by-Step Guide to Exporting Data to Text Files

Authored by

Mohammed looti

November 14, 2025

Introduction to Data Export in SAS

In the vast landscape of data management and advanced statistical analysis, the ability to move data efficiently between different computing environments is paramount. SAS, globally recognized as a premier software suite for sophisticated analytics, provides robust tools for handling and manipulating diverse datasets. A frequent and essential requirement in many analytical workflows is exporting structured data from the proprietary SAS environment into a simple, universally compatible text file format. This critical process facilitates seamless data exchange with partners who may not use SAS licenses, supports integration with non-SAS systems, and allows for the creation of easily readable output for archival or manual review purposes.

The primary mechanism for achieving this necessary export functionality in SAS is the **PROC EXPORT** statement. This high-efficiency procedure is the standard utility for converting SAS datasets into various external formats, including comma-separated values (CSV), tab-delimited files, spreadsheets, and even external database tables. Utilizing **PROC EXPORT** ensures that the data structure and integrity are meticulously maintained throughout the transformation process, offering both reliability and efficiency for mission-critical operations.

This comprehensive guide is structured to explore the full capabilities of **PROC EXPORT**. We will systematically break down its fundamental syntax, detail the essential arguments required for successful execution, and provide clear, practical examples. These demonstrations will illustrate how to transfer a dataset into a standard text file, covering both simple default exports and highly customized configurations necessary to meet specific formatting requirements, such as altering the field delimiter or suppressing column names.

Understanding the `PROC EXPORT` Statement

The **PROC EXPORT** statement serves as a foundational utility within the SAS programming environment. Its primary function is to enable the controlled output of data stored within a SAS dataset to an external file location. This procedure is expertly designed to simplify the complex tasks associated with data conversion and file formatting, ensuring that the exported data maintains the correct structure and encoding required by the target application or receiving system.

The core syntax required to execute **PROC EXPORT** is designed to be straightforward, making it accessible even to novice SAS users. Successful execution typically depends on specifying three critical components: identifying the source dataset (DATA), defining the full path for the resulting output file (OUTFILE), and specifying the desired Database Management System (DBMS) type. When the goal is to export data to a standard text file, the **`DBMS=TAB`** option is frequently used to generate tab-delimited output, which is a highly portable and widely accepted format for data exchange.

The standard structure below illustrates the essential code components for efficiently exporting data to a text file. This template incorporates the crucial arguments that govern the source, destination, format, and behavior of the export process, specifically leveraging the `**REPLACE**` option to seamlessly manage potential existing files:

```
/*export data to file called my_data.txt*/
proc export data=my_data
outfile="/home/u13181/my_data.txt"
dbms=tab
replace;
run;
```

This code snippet clearly defines the necessary parameters: the input dataset (`**DATA=my_data**`), the absolute path for the output text file (`**OUTFILE="..."**`), the file format specification (`**DBMS=TAB**`), and the instruction to overwrite any existing file at that location (`**REPLACE**`). Developing a complete understanding of how each of these arguments functions is essential for conducting successful, repeatable, and precisely tailored data export operations within the SAS environment.

Key Arguments for Customizing `PROC EXPORT`

To fully harness the capabilities of **PROC EXPORT**, analysts must thoroughly understand the function of its core and optional arguments. These parameters provide granular control over the source data, the destination path, and the final output format, guaranteeing that the resulting text file adheres to all external specifications required by other systems.

`DATA`: This serves as the sole mandatory positional argument, crucial for identifying the SAS dataset targeted for conversion. For instance, declaring `**DATA=my_data**` directs the procedure to utilize the dataset named `**my_data**` currently residing in your active SAS library or session. Prior verification of the dataset's existence and accessibility is a necessary precursor to initiating the export procedure.

`OUTFILE`: This argument dictates the exact location and file name of the resulting external file. A clear declaration, such as `**OUTFILE="/home/u13181/my_data.txt"**`, instructs SAS to create the specified file within the designated directory. It is vital to use either an absolute path or a verified relative path and to ensure the SAS execution environment possesses the necessary write permissions for the target location on the file system.

`DBMS`: The DBMS argument defines the desired output file format. For generating a generic delimited text file, the standard choice is `**DBMS=TAB**`, which yields a tab-delimited structure.

While options like `**CSV**` (for comma-separated files) are available, `**TAB**` is widely considered the default specification for versatile text file output when no custom delimiter is explicitly overriding it.

`REPLACE`: Highly recommended for automated or repeated processes, `**REPLACE**` instructs SAS to automatically overwrite the output file if a file with the same name already exists at the `**OUTFILE**` path. If this option is omitted and the file is present, SAS will terminate the procedure with an error. Users must employ `**REPLACE**` cautiously to prevent unintentional data loss or modification of critical archived files.

`DELIMITER`: This optional yet crucial argument allows users to specify a custom delimiter character, effectively overriding the default separation behavior set by `**DBMS**`. Common custom delimiters include semicolons (`**;**`), commas (`**,**`), or vertical bars (`**|**`). For example, using `**DELIMITER=";"**` produces a semicolon-separated file, a format often mandatory when exchanging data with international systems or specific receiving platforms.

`PUTNAMES`: This argument controls whether the variable names--the header row--are included in the initial line of the exported text file. By default, `**PUTNAMES=YES**` is active, ensuring headers are present. Conversely, setting `**PUTNAMES=NO**` results in the export of only the raw data values, a requirement frequently imposed by systems that expect pure, unformatted record input without column labels.

Example 1: Exporting a Dataset to a Tab-Delimited Text File

To illustrate the utility of **PROC EXPORT** in its most straightforward application, we will first execute an export using default settings. This generates a standard tab-delimited text file that includes the variable names as the header row. This method represents the most common and versatile approach for general data exchange requirements.

We begin by establishing a sample dataset named `**my_data**`. This synthetic dataset contains typical numerical metrics, simulating player performance with variables such as rating, points, assists, and rebounds. This structured data will serve as the source material for our export operation, ensuring we have verifiable content ready for processing.

```
/*create dataset*/
data my_data;
input rating points assists rebounds;
datalines;
90 25 5 11
85 20 7 8
82 14 7 10
88 16 8 6
```

```
94 27 5 6
90 20 7 9
76 12 6 6
75 15 9 10
87 14 9 10
86 19 5 7
;
run;

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

Executing the `**PROC PRINT**` command allows for an immediate internal review of the newly created `**my_data**` dataset within the SAS output environment. This confirmation step is crucial for validating that the source data is correctly formatted and prepared for the subsequent export procedure, as illustrated in the image below:

Obs	rating	points	assists	rebounds
1	90	25	5	11
2	85	20	7	8
3	82	14	7	10
4	88	16	8	6
5	94	27	5	6
6	90	20	7	9
7	76	12	6	6
8	75	15	9	10
9	87	14	9	10
10	86	19	5	7

To initiate the export of `**my_data**` to the external text file named `**my_data.txt**`, we employ the basic structure of **PROC EXPORT**. We rely on the `**DBMS=TAB**` setting to ensure proper tab-delimited formatting, which inherently includes the `**PUTNAMES=YES**` behavior by default:

```
/*export dataset*/
proc export data=my_data
outfile="/home/u13181/my_data.txt"
dbms=tab
```

```
replace;
run;
```

Upon successful execution, the specified file, `**my_data.txt**`, is created at the designated path. When inspected using a standard text editor or spreadsheet application, the file confirms that the data has been structured with tab characters separating fields, and the variable names are correctly positioned on the first line. This validates the success of the default export procedure and ensures the output is ready for immediate use in other applications.

Example 2: Customizing Text File Export using Delimiters and Headers

While default export settings are often sufficient, complex integration scenarios frequently demand precise control over the output file's formatting. The flexibility of **PROC EXPORT** allows for this customization through the specialized `**DELIMITER**` and `**PUTNAMES**` arguments. This example focuses on generating a file that utilizes a custom field separator and intentionally omits the column headers, requirements common when integrating with legacy systems or specific data ingestion pipelines.

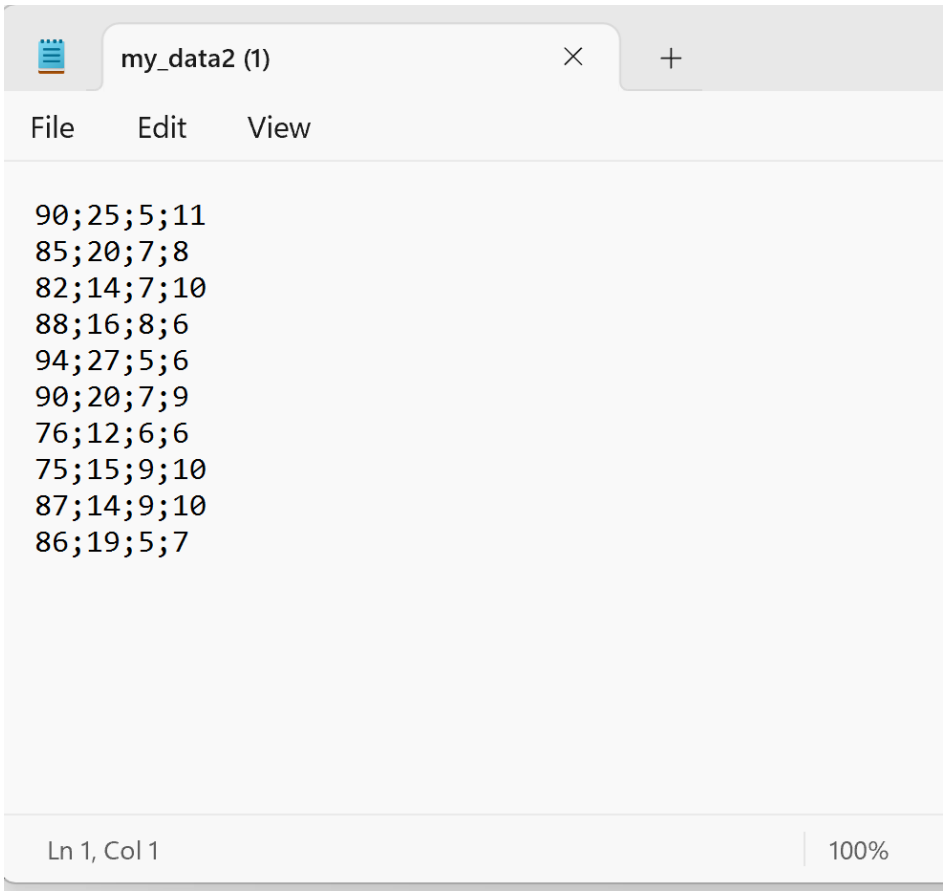
The `**DELIMITER**` argument empowers the analyst to specify any character as the field separator, moving beyond the default tab or comma. For instance, in contexts where commas serve as decimal markers (prevalent in certain European regions), a semicolon (`;`) is often preferred to prevent parsing conflicts. For this demonstration, we implement **DELIMITER=";"** to create a semicolon-separated file. Simultaneously, the `**PUTNAMES=NO**` argument is employed to instruct SAS to entirely suppress the inclusion of the header row, ensuring the exported file contains only raw data records.

We apply these custom specifications to export the `**my_data**` dataset to a new file named `**my_data2.txt**`:

```
/*export dataset*/
proc export data=my_data
outfile="/home/u13181/my_data2.txt"
dbms=tab
replace;
delimiter=";";
putnames=NO;
run;
```

Even though `**DBMS=TAB**` is retained, the explicit DELIMITER option overrides the default tab-delimited character setting. After the code executes, inspecting `**my_data2.txt**` confirms that the

desired transformation has successfully occurred. The file now uses semicolons as field separators, and the entire header row is absent, starting immediately with the first record of the data, as demonstrated in the visual output below. This level of precision is invaluable for complex data export requirements.



Notice that the header row has been removed, and the values within each row are separated by semicolons instead of tabs. This level of customization ensures adherence to precise data exchange specifications.

Note: You can find the complete documentation for the **PROC EXPORT** statement here.

Best Practices and Troubleshooting Data Export

Ensuring successful and reliable data export operations from SAS requires adhering to several critical best practices and maintaining a systematic approach to troubleshooting. The first and most essential step is always to verify the integrity and structure of the exported text file. This involves opening the file in a suitable application, such as a specialized text editor or spreadsheet program, to visually confirm that the data is correctly separated, the header row is present (or absent) as intended, and that data encoding is preserved correctly.

Common issues encountered during the export process often revolve around three core technical areas. Firstly, **file paths and write permissions**: A frequent error occurs when the specified `**OUTFILE**` path is either syntactically incorrect or if the SAS process lacks the necessary operating system permissions to write to that directory. Secondly, **Delimiter Conflicts**: If the chosen delimiter character (e.g., a comma or semicolon) happens to exist within the data values themselves, the receiving system will incorrectly parse the columns, leading to corrupted records. Careful selection of an unambiguous delimiter is paramount. Finally, **Data Type Handling**: While SAS is proficient at converting most standard data types, complex or specialized formats may require pre-processing using `**PROC SQL**` or `**DATA steps**` to ensure they are correctly represented as plain text before the export procedure runs.

To mitigate risk, especially when working with extensive or highly sensitive data, it is strongly recommended to adopt an iterative testing methodology. Start by testing your **PROC EXPORT** statements on a small, representative subset of your dataset. This approach allows for rapid identification and correction of any syntax or formatting errors before committing to a resource-intensive full data export, thereby significantly enhancing efficiency and maintaining data quality assurance.

Further Resources for SAS Data Management

Mastering data export represents just one critical component of comprehensive data management within the SAS framework. Analysts must possess equally strong competencies in importing, transforming, and manipulating data to execute complete and insightful data analysis workflows. We strongly encourage users to continue expanding their expertise by exploring the vast array of additional functionalities and advanced procedures available within the SAS software suite, thereby strengthening their overall data science capabilities.

For those seeking to delve deeper into SAS data handling, the official SAS documentation serves as the definitive resource, offering meticulously detailed guides and extensive examples for every procedure, including advanced statistical modeling and complex data manipulation techniques. Furthermore, numerous high-quality tutorials are available online, providing step-by-step guidance on common tasks ranging from efficient data import strategies to the application of sophisticated statistical methods.

Understanding how to reverse this process--importing external files back into SAS--is an essential complementary skill:

How to Import Text Files into SAS