Export Data from SAS to CSV File (With Examples)

Authored by

Mohammed looti

November 1, 2025

The efficient transfer of data across diverse computing environments is a cornerstone of effective modern data analysis workflows. For professionals operating within the SAS ecosystem, the need to extract structured data and transform it into a universally readable format, such as CSV (Comma Separated Values), is a frequent and critical requirement. Fortunately, SAS provides an extremely robust and straightforward mechanism for this purpose: the PROC EXPORT procedure. This powerful tool allows users to quickly and accurately generate external files from existing SAS datasets, ensuring seamless interoperability.

This comprehensive guide is designed to dissect the structure and application of **PROC EXPORT**. We will demonstrate how to leverage its full functionality for various data export requirements, ranging from the simplest default settings suitable for quick transfers to highly customized outputs necessary for integration with specialized downstream applications. By mastering this procedure, data analysts can significantly enhance the efficiency and reliability of their data interchange processes.

Introduction to Data Export in SAS

Data residing within a SAS dataset often serves as the input source for external applications, which might include spreadsheet software like Microsoft Excel, relational database management systems, or specialized statistical packages. The **CSV format** is widely recognized as the industry standard for plain text data exchange, primarily due to its inherent simplicity, minimal overhead, and near-universal compatibility across platforms and systems.

While SAS offers several methods for exporting data, **PROC EXPORT** stands out as the preferred choice for most users. Its popularity stems from its exceptional ease of use and its ability to effortlessly handle various file structures, including delimited files like CSV. The procedure effectively streamlines the entire export process by automatically managing file handling, facilitating necessary variable format conversions, and simplifying the specification of the output file path through a concise, parameter-driven syntax.

Achieving proficiency with **PROC EXPORT** enables analysts to guarantee the seamless integration of their sophisticated SAS-processed data into any required downstream application. This capability is vital for maintaining high levels of data integrity throughout the workflow and maximizing overall operational efficiency, transforming complex data transfers into routine, reliable tasks.

Understanding the PROC EXPORT Statement Structure

The fundamental functionality required to successfully generate a CSV file from SAS data is encapsulated within the basic syntax of the **PROC EXPORT** statement. Successful execution of this procedure necessitates the explicit definition of three critical components: the specific source

dataset being exported, the exact location and filename for the output file, and the designated Database Management System (DBMS) type, which specifies the desired file format.

To initiate a rapid and accurate data transfer, the **PROC EXPORT** command must follow a precise structure. The essential syntax below illustrates this core framework, designed to export a dataset to an external file named data.csv:

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

The definitive action that initiates the data transfer is the execution of the **RUN** statement, which commits the specified SAS dataset to the external file path designated by the user in the OUTFILE parameter. A thorough understanding of each option within this procedure is absolutely essential for achieving successful and predictable export operations.

Here's a detailed breakdown of what each essential parameter achieves within the procedure, clarifying their roles in managing the data transfer:

data: This parameter specifies the name of the source SAS dataset that is intended for export. It is an argument that is strictly mandatory for the procedure to function.
outfile: This defines the complete, absolute file path and the filename where the resulting CSV file will be saved. Users must ensure that the SAS system process possesses the necessary write permissions for the specified directory location.
dbms: This indicates the file format or database management system type to be utilized for the export operation. For standard comma-separated files, this parameter must be explicitly set to **CSV**.
replace: While optional, this statement is highly recommended. It serves as an instruction to SAS to automatically overwrite the output file if a file sharing the same name already exists at the specified OUTFILE location, preventing execution errors.

These four core elements establish the fundamental framework for any data export operation executed using **PROC EXPORT**. The following section will explore how to extend this foundational functionality using additional options to meet specific formatting requirements.

Key Parameters for Customizing CSV Output

While the basic syntax of **PROC EXPORT** effectively handles standard data transfers, real-world

analytical requirements frequently necessitate specific formatting adjustments to ensure compatibility with diverse target systems. Fortunately, the procedure offers several powerful statements that grant fine-grained control over the output file's structure, including the capability to select alternative field separators and manage the inclusion of variable names.

One of the most powerful and frequently used customizations involves the **DELIMITER** option. Although the acronym CSV implicitly suggests a comma separator, specific regional settings or certain target database systems may require alternative field separators, such as semicolons, tabs, or pipe characters. The **DELIMITER** statement provides the flexibility to define any character as the field separator, making the output highly adaptable to various international data standards and specialized application inputs.

Another critical customization involves controlling the presence of the header row. By default, **PROC EXPORT** automatically writes the SAS variable names as the very first row in the CSV file. If the receiving system is designed to process data without a header, or if it requires a distinct set of column labels defined externally, the **PUTNAMES=NO** option must be utilized. This option actively suppresses the writing of the variable names, ensuring compatibility with systems that strictly expect pure data rows beginning from the first line.

A deep understanding and proper application of these customizing parameters are essential. They ensure that the exported file is not only generated correctly and efficiently but is also immediately usable by the receiving application, eliminating the need for time-consuming manual modifications or reformatting steps upon arrival.

Example 1: Standard Export Using Default Settings

To clearly illustrate the fundamental application of PROC EXPORT, we begin with a practical, simple scenario. We will first construct a small SAS dataset and then proceed to export it using the procedure's default CSV configuration. This approach is optimal when the standard comma delimiter is acceptable for the target system and a standard header row containing variable names is desired.

The first prerequisite is to define the dataset that we intend to transfer. We achieve this by utilizing a standard **DATA** step in conjunction with the **DATALINES** statement to quickly generate a small, manageable set of observations:

```
/*create dataset*/
data my_data;
input A B C;
datalines;
1 4 76
```

```
2 3 49
2 3 85
4 5 88
2 2 90
4 6 78
5 9 80
;
run;

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

The code above successfully generates a temporary dataset named **my_data**, which contains three distinct variables (A, B, and C) and seven total observations. Executing the subsequent **PROC PRINT** step allows us to view and confirm the exact structure and content of the dataset before proceeding with the critical export operation:

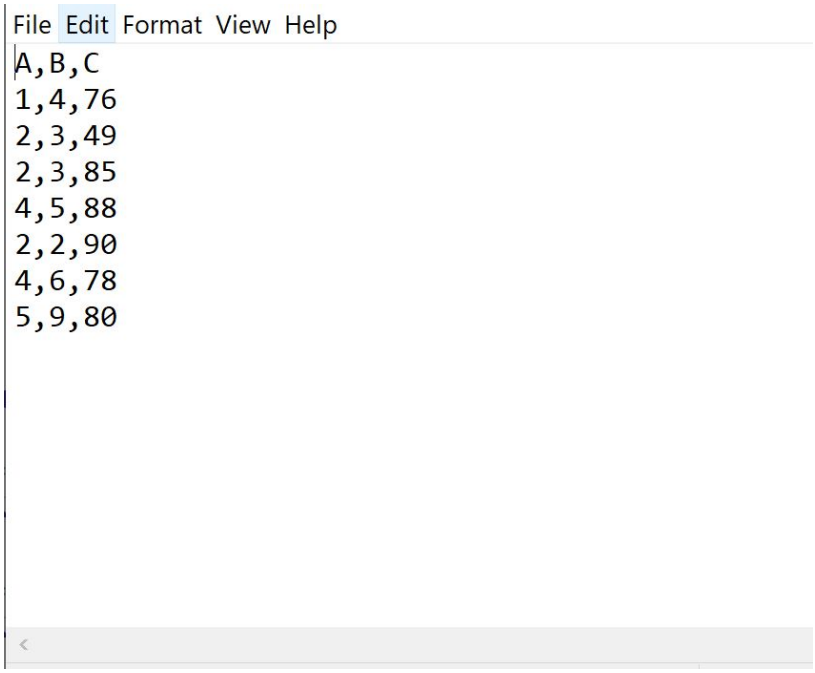
Obs	A	B	C
1	1	4	76
2	2	3	49
3	2	3	85
4	4	5	88
5	2	2	90
6	4	6	78
7	5	9	80

With the source data verified, we can now proceed to export this dataset to the desired CSV file. We use the following concise code block, ensuring that we specify the absolute output path and include the **REPLACE** option to gracefully handle any pre-existing file conflicts at that location:

```
/*export dataset*/
proc export data=my_data
outfile="/home/u13181/data.csv"
dbms=csv
replace;
run;
```

Once the procedure executes without error, the file **data.csv** is created precisely at the specified output location. By navigating to the file system and opening the exported file in a suitable application, such as a text editor or a spreadsheet program, we can visually confirm that the original data structure has been perfectly preserved, complete with column headers and fields accurately separated by commas.

The following image confirms the successful structure of the exported file when viewed externally:



As this illustration demonstrates, the data contained within the CSV output file is an exact, structured match to the source dataset within SAS, confirming the flawless execution using the default **PROC EXPORT** settings.

Example 2: Custom Delimiters and Header Suppression

There are many scenarios where standard comma separation is insufficient. For instance, exporting data to certain applications, particularly those common in European contexts, may require the semicolon (;) as the mandatory field separator. Furthermore, some target systems demand a strictly data-only file, necessitating the complete omission of the header row. In these advanced cases, rigorous customization of the export parameters becomes absolutely necessary to meet strict external data conventions.

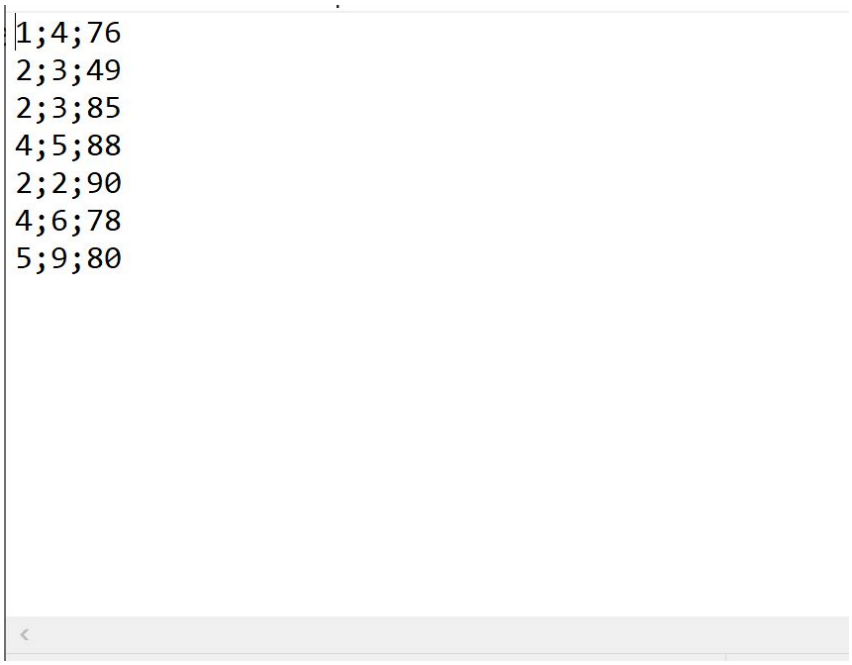
The code presented below demonstrates how to precisely export the **my_data** SAS dataset into a CSV file while implementing two significant customizations: utilizing a semi-colon as the primary delimiter and simultaneously omitting the header row entirely from the output file.

These specialized requirements are fulfilled by incorporating the **DELIMITER=** and **PUTNAMES=NO** options directly within the **PROC EXPORT** block, overriding the procedural defaults:

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

The inclusion of the **DELIMITER=";"** statement explicitly overrides the standard comma setting, ensuring every field is accurately separated by a semicolon. More critically, the **PUTNAMES=NO** statement prevents the inclusion of the SAS variable names (A, B, C) in the crucial first line of the output file, resulting in a data-only structure.

Observing the resulting output file confirms that these custom parameters have been correctly applied:



As clearly demonstrated, the header row has been successfully removed, and all data values are now separated by semi-colons instead of the default commas. This capability for fine-tuned customization is essential for ensuring that the exported file precisely adheres to stringent external

data import specifications.

Troubleshooting and Best Practices for SAS Data Export

Although PROC EXPORT is generally considered a highly stable and robust procedure, users may occasionally encounter operational issues. These typically revolve around file access permissions, incorrect pathing, or complexities related to character encoding. Adopting a set of rigorous best practices is crucial for proactively preventing these common pitfalls and guaranteeing consistently reliable data transfer performance.

The single most frequent point of failure relates to the file path specified in the **OUTFILE** parameter. If the SAS execution environment is running on a remote server (common in setups like SAS Viya or SAS Enterprise Guide), the path must reference a location that is fully accessible and writable by the SAS server process itself, not necessarily the local machine where the user is executing the code. Incorrect paths or, more commonly, a lack of appropriate write permissions for the SAS service account are the primary causes of execution failures and error messages.

Furthermore, handling character data, particularly when dealing with non-English characters or specialized symbols, demands careful attention to character encoding settings. SAS typically operates using a specific default encoding (e.g., Latin1 or UTF-8). If the destination system expects a different encoding standard, users must utilize the **ENCODING=** option within **PROC EXPORT** or ensure that the SAS session encoding is adjusted beforehand. Failure to match encoding standards will inevitably lead to corrupted characters or data loss in the final output CSV file.

Finally, when managing exceptionally large datasets, processing time can become a significant performance bottleneck. While **PROC EXPORT** is highly optimized for efficiency, for truly massive datasets--those involving millions or even billions of rows--alternative strategies should be considered. This might involve utilizing specialized SAS Access interfaces if the target is a database, or leveraging high-performance computing capabilities if they are integrated within your enterprise SAS environment to drastically optimize data throughput.

Summary and Conclusion

The **PROC EXPORT** statement is an essential and indispensable utility for every SAS user who requires the exchange of processed data with external analytical or operational systems. Its inherent flexibility, driven by powerful customization parameters such as **DELIMITER** and **PUTNAMES**, ensures that the resulting CSV file can be tailored to meet virtually any structural or regional requirement imposed by the receiving application. By achieving proficiency in both the core syntax and the effective application of these customization options, analysts can maintain optimal data flow efficiency, accuracy, and reliability across all their complex analytical workflows.

To further expand your skills in the SAS programming language, the following tutorials detail how to perform other common and critical data manipulation tasks: