How to Easily Export Data from SAS to CSV

Authored by

stats writer

December 1, 2025

The ability to efficiently move data between different systems is fundamental in modern data management and analysis. When working within the SAS environment, analysts frequently need to share results or transfer processed information to systems that rely on standardized, text-based formats. The most common and universally accepted format for this purpose is the CSV (Comma-Separated Values) file. This format offers unparalleled interoperability, making it essential knowledge for any professional using SAS for statistical computing.

Fortunately, SAS provides an extremely robust and straightforward mechanism for this task: the **EXPORT procedure**, commonly invoked using the statement PROC EXPORT. This powerful procedure is specifically designed to facilitate the rapid and reliable transfer of a SAS data set into external file formats, including database tables, spreadsheets, and, most importantly here, delimited text files. By mastering PROC EXPORT, users can specify not only the source data and destination path but also customize crucial elements like the column delimiter and the presence of header rows, ensuring the output file is perfectly tailored for the receiving application.

This comprehensive guide will detail the exact syntax and necessary steps required to utilize PROC EXPORT for converting internal SAS structures into external CSV files. We will explore the core parameters, walk through practical examples demonstrating both default and customized export settings, and provide a deep dive into how each option influences the final output, thereby enabling the generation of clean, valid, and easily readable data files for any external system or user.

Understanding the SAS EXPORT Procedure: Syntax and Functionality

The primary tool for exporting data out of the SAS system is the **proc export** statement. This procedure acts as a bridge, reading a specified internal data set and transforming it according to predefined parameters into an external file format, such as CSV. The efficiency of PROC EXPORT lies in its simplicity and directness, requiring only a few mandatory arguments to perform a complete data transfer operation, making it ideal for both interactive use and integration into larger automated scripts.

To successfully execute a data export, the procedure must be supplied with four critical pieces of information. First, the data set name must be provided via the `DATA=` option, identifying the source table within the SAS library. Second, the full path and filename for the resulting output file must be specified using the `OUTFILE=` option. Third, the type of output file must be declared using the `DBMS=` option, which is essential for ensuring that SAS structures the output correctly--for standard comma-separated files, this will always be `CSV`. Finally, while not strictly mandatory, the `REPLACE` option is highly recommended to instruct SAS to overwrite the destination file if it already exists, thus avoiding execution errors related to file conflicts.

The fundamental syntax structure for using **proc export** to generate a CSV file involves a clear

block of code that defines these parameters, concluding with the necessary `RUN` statement to execute the procedure. Understanding this basic structure is the first step toward mastering data output in SAS, as nearly all subsequent customization options build upon this core framework. This standardized approach ensures consistency and reliability across different analytical projects and environments.

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

Core Parameters of PROC EXPORT for CSV Generation

Each keyword within the PROC EXPORT syntax serves a specific, vital function in the data transfer process. Detailed understanding of these parameters ensures that the export operation runs smoothly and produces the desired output file structure. Missing or incorrectly specified parameters can lead to errors, corrupted output files, or data loss, underscoring the necessity of precision when defining the procedure arguments.

The following list details the purpose of each key parameter used in the basic **proc export** structure when generating a CSV file, providing clarity on how the procedure interprets these instructions:

data: This mandatory option specifies the exact name of the source data set located within the active SAS library that the user wishes to export. If the dataset resides in a specific library (e.g., `LIBREF.DATASET`), the full two-level name must be used.
outfile: This parameter dictates the complete file path, including the desired filename and extension (typically `.csv`), where the resulting exported data file will be saved on the operating system's file structure. Security and permissions must be managed by the user to ensure SAS has write access to this location.
dbms: Standing for Database Management System, this parameter instructs SAS on the internal format driver to use for the output. For a standard delimited file, setting `dbms=csv` ensures that the output is formatted using comma separators and appropriate text qualifiers, which is the primary driver for creating a valid CSV file.
replace: This optional yet highly recommended statement informs SAS that if a file already exists at the specified `OUTFILE` location, the procedure should automatically overwrite it without prompting or halting execution. Omission of this option when a destination file exists will typically result in a non-fatal error or warning, preventing the export from completing.

Detailed Walkthrough: Exporting Data Using Default Settings (Example 1)

To illustrate the practical application of PROC EXPORT, we will first establish a sample data set within the SAS environment. This dataset, named `my_data`, will contain three variables (A, B, C) and seven observations, representing typical tabular data that would need to be shared externally. The creation of the source data uses the fundamental SAS DATA step, relying on the `INPUT` and `DATALINES` statements for rapid data entry, a common practice for demonstrating procedures with small examples.

Once the `my_data` data set is successfully created and compiled, it resides in the temporary work library. To confirm its structure and content before export, we employ the `PROC PRINT` procedure, which generates a report showing the variables and their corresponding values. This verification step is crucial in production environments to ensure that the correct data is being prepared for transfer, preventing the accidental export of incomplete or incorrect information.

The code below demonstrates this preliminary stage, establishing the foundation upon which the export operation will be built. This is the starting point for utilizing PROC EXPORT, ensuring that the data structure is ready and validated before any external file manipulation occurs. The visual representation (image following the code) confirms the structure of the data as viewed internally by SAS.

```
/*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;
```

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

Executing the Default CSV Export

With the source data verified, we proceed to the core task: exporting the `my_data` data set to a CSV file named `data.csv`. In this first example, we utilize the simplest form of PROC EXPORT, relying entirely on the default settings inherited when `DBMS=CSV` is specified. These defaults typically include using a comma (,) as the field delimiter and automatically including the variable names from the SAS data set as the header row in the output file.

The following code block executes the export operation, clearly specifying the data source (`data=my_data`), the full destination path (`outfile="/home/u13181/data.csv"`), and the file type (`dbms=csv`). The inclusion of `replace` ensures that the file is created successfully, even if a previous version exists. Upon successful execution of the `RUN` statement, the operating system file system will contain the new `data.csv` file at the specified path, ready for inspection or use by other software applications.

The subsequent step involves navigating outside of the SAS environment--often using a file explorer or a text editor--to verify the contents of the newly created CSV file. As demonstrated by the image below, the output file contains the three variable names (A, B, C) as the first row, followed by the seven rows of numerical data, with all values separated by commas. This confirms that the default export settings successfully mirrored the structure of the internal SAS data set.

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

I can then navigate to the location on my computer where I exported the file and view it:

The data in the CSV file matches the data set from SAS exactly, confirming successful extraction using default parameters.

Advanced Configuration: Customizing CSV Output (Example 2)

While the default comma delimiter and header row are suitable for many applications, there are frequent scenarios, particularly when integrating data with legacy systems or specialized analytical tools, where customization is necessary. PROC EXPORT offers advanced options that allow granular control over the output format, ensuring compatibility with diverse requirements. Two of the most commonly customized elements are the field delimiter and the inclusion of variable names as the header row.

In certain European data standards or database transfer protocols, the semicolon (;) is preferred over the comma for separating fields, especially when dealing with data that might contain commas within text strings. To accommodate this, PROC EXPORT utilizes the `DELIMITER=` statement. By setting `DELIMITER=";"`, SAS ensures that every variable value in the output file is separated by a semicolon instead of the default comma, significantly enhancing interoperability in locale-specific contexts.

Furthermore, many data loading processes require raw data without variable names, expecting only the data values themselves. The `PUTNAMES=` option manages the inclusion of the header row. By default, this option is set to `YES`. To suppress the header row containing the SAS variable

names, the user simply sets `PUTNAMES=NO` within the PROC EXPORT step. These two options--`DELIMITER` and `PUTNAMES`--demonstrate the flexibility of the procedure in tailoring the CSV output to exact specifications, moving beyond the simple default configuration.

Controlling Headers and Delimiters for Interoperability

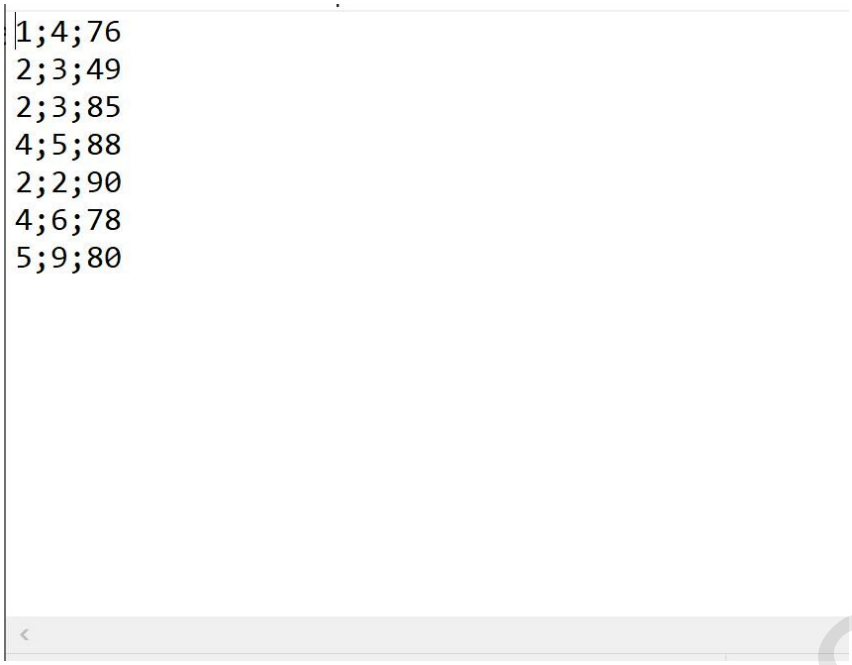
The following example demonstrates how to apply these custom settings to the export operation, using the same `my_data` data set established in Example 1. The primary structural components remain identical (`DATA`, `OUTFILE`, `DBMS`, `REPLACE`), but two new parameters are introduced to enforce specific formatting rules necessary for external integration. These customized exports are crucial when dealing with external DBMS systems or applications that have strict input formatting requirements.

In the code block provided, observe the inclusion of `delimiter=";"` and `putnames=NO`. This combination explicitly instructs SAS to perform a specialized export: fields must be separated by a semicolon, and the first row, which traditionally holds the column names, must be omitted entirely. This results in a cleaner data file consisting only of the raw values, separated by the non-default delimiter.

Reviewing the exported file confirms the successful application of these custom settings. As shown in the visual verification, the column headers (A, B, C) are missing, and the numerical values themselves are now visibly separated by semicolons rather than commas. This outcome validates the use of the advanced options and confirms that PROC EXPORT is highly adaptable for meeting complex data interoperability needs beyond basic CSV creation.

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

I can then navigate to the location on my computer where I exported the file and view it:



Notice that the header row has been removed and the values are separated by semi-colons instead of commas, confirming the successful use of custom options.

Summary of Best Practices and Troubleshooting Tips

While PROC EXPORT is generally reliable, adopting certain best practices ensures smoother operation and minimizes potential troubleshooting issues, especially when dealing with large data sets or complex environments. Always ensure the output path specified in `OUTFILE=` is fully qualified and that the SAS session possesses the necessary write permissions for that directory. Permission errors are among the most common causes of failed exports, often resulting in obscure error messages in the SAS log rather than direct file access warnings.

When customizing the export, pay careful attention to the `DELIMITER=` option. While commas and semicolons are standard, any single character can theoretically be used as a delimiter. However, choosing a character that does not appear anywhere within the actual data values is crucial. If the chosen delimiter exists in a data field, the output file will be improperly structured, leading to incorrect parsing when imported by external systems. If data integrity is a concern, consider using the `TEXTQUALIFIER=` option, which instructs SAS to enclose text strings in quotes, thus protecting them from being misinterpreted by the delimiter.

Finally, always confirm the structure of the exported CSV file after execution. While SAS reports successful execution, a quick inspection using a standard text editor confirms that the headers are included or suppressed as intended, and the fields are correctly separated by the expected delimiter. This final validation step guarantees that the data is ready for seamless integration into

its destination system, completing the critical data transfer pipeline efficiently and accurately.

Further SAS Resources

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