Learning SAS: Mastering PROC IMPORT for Data Integration

Authored by

Mohammed looti

November 14, 2025

In modern data science and statistical computing environments, the foundational step for any meaningful analysis is the successful integration of external data. Without reliable methods to ingest information from disparate sources, sophisticated modeling remains impossible. For users of SAS, the **PROC IMPORT** statement stands as the essential utility for this critical task. This powerful procedure allows analysts to seamlessly convert data from various external file formats--including spreadsheets, databases, and text files--into native SAS datasets. This guide provides a comprehensive walkthrough, detailing how to leverage **PROC IMPORT** effectively to ensure all your data integration projects are executed with precision and success.

Deconstructing the `PROC IMPORT` Syntax Fundamentals

The design philosophy behind **PROC IMPORT** prioritizes both simplicity and operational efficiency. Its primary function is to read an external data file, interpret its structure, and translate its contents into a standard SAS dataset. This transformation makes the data instantly available for subsequent use across all SAS procedures, from data manipulation to advanced statistical modeling. A deep understanding of the fundamental structure of this statement is indispensable for robust data management within the vast SAS ecosystem.

The basic syntax of **PROC IMPORT** is highly intuitive, relying on a small set of mandatory and optional arguments that govern the entire import process. These arguments act as precise instructions, informing SAS exactly how to locate, read, parse, and store the source file. Correctly specifying these arguments is paramount to maintaining data integrity and ensuring the resulting dataset accurately reflects the external source structure.

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

To clarify the mechanism of this procedure, we must examine the specific role played by each essential argument within the syntax block above. These parameters dictate the output location, the source file's identity, the required parsing engine, and how existing data conflicts are resolved.

out: This vital argument designates the name of the new SAS dataset that will be generated upon successful import. For example, using `out=my_data` instructs SAS to store the imported data under the name `my_data`. If a two-level name (e.g., `mylib.my_data`) is not provided, the dataset is automatically stored in the temporary **WORK** library, which is erased when the SAS session ends.
datafile: This crucial argument specifies the complete file path leading to the external data source.

It is mandatory that this path be accurate and complete, enabling SAS to precisely locate the source file on the system. For instance, `datafile="/home/u13181/my_data.csv"` directs the procedure to a specific CSV file within a designated directory structure.

dbms: Arguably the most critical parameter, **dbms** defines the format of the external file. It instructs SAS which database management system or file type driver to utilize for reading and interpreting the raw data. Key values include `csv` for comma-separated values, `xlsx` for Microsoft Excel files, and `dlm` for generic delimited text files. Choosing the correct **dbms** value ensures that the appropriate parsing logic is applied.

replace: The inclusion of the `replace` option grants SAS permission to overwrite any existing SAS dataset that shares the same name within the specified library. This is extremely valuable for data updates or when iterating on an import script. If this option is omitted and a dataset with the same name already exists, SAS will typically issue an error message and terminate the procedure execution, safeguarding existing data.

getnames: This parameter controls how SAS handles the very first row of the external data file. When set to `YES` (as in `getnames=YES`), SAS interprets the values in the first row as the variable names (headers) for the new SAS dataset. Conversely, if the first row contains actual data values, setting this option to `NO` is essential to ensure that data is read starting from the first line.

Leveraging the `DBMS` Option for Multi-Format Data Integration

The unparalleled flexibility of **PROC IMPORT** is largely attributed to the versatility provided by its **dbms** argument. This single option empowers the user to import virtually every common type of external data file into the SAS environment simply by modifying its value to match the format of the source file. This capability makes **PROC IMPORT** an indispensable and highly functional tool for data professionals who regularly interface with diverse, multi-format data sources.

The crucial prerequisite for a successful import operation is the accurate identification of the input file's format and the subsequent specification of the corresponding **dbms** value. SAS maintains specialized drivers, often referred to as engines, tailored to handle a wide range of file types. These drivers ensure rigorous parsing and accurate data conversion into native SAS formats. Failure to correctly identify the file type will inevitably lead to processing errors, data interpretation issues, or corrupted variable definitions in the final SAS dataset.

To illustrate the power of this argument, we detail the necessary **dbms** specification for three of the most frequently encountered file types, highlighting the specific characteristics SAS is instructed to handle:

To import a CSV file (Comma-Separated Values), the required specification is **`dbms=csv`**. This command explicitly directs SAS to anticipate data fields separated by commas, automatically applying the default delimiter recognition logic for this ubiquitous format.

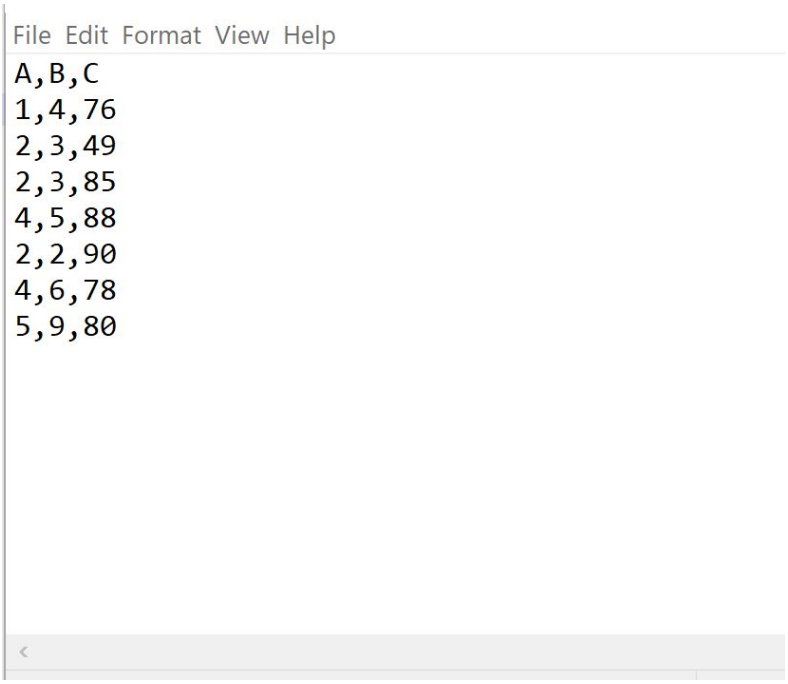
For importing a Microsoft Excel file (which typically uses `.xlsx` or older `.xls` extensions), the correct specification is **`dbms=xlsx`**. This leverages SAS's dedicated Excel engine, allowing it to navigate complex spreadsheet structures, including handling multiple worksheets, merged cells, and complex cell formatting during the data extraction process.
When integrating a generic Delimited Text file where values are separated by a non-standard character (e.g., tabs, pipes, or semicolons), the user must specify **`dbms=dlm`**. This option provides essential flexibility, requiring the analyst to further define the exact delimiter character using the supplemental `DELIMITER=` option within the procedure block.

The following practical examples transition from theoretical syntax to applied coding, demonstrating the effective deployment of **PROC IMPORT** for these three common file types. Each illustration provides clear code blocks and showcases the resulting SAS dataset to confirm successful data conversion.

Example 1: Importing Tabular Data from a CSV File

Comma-Separated Values (CSV) files represent one of the most widely adopted and fundamental formats for the interchange of tabular data. Their inherent simplicity--being purely plain text--ensures incredible versatility and compatibility across virtually all operating systems and data analysis platforms. We will now walk through a practical scenario requiring the import of a CSV file named **my_data.csv** into the SAS environment for immediate statistical analysis.

Our external source file, **my_data.csv**, contains a small but representative dataset. It is important to note that the very first row of this file serves as the column headers, which must be recognized as variable names by SAS.



To successfully load this data and create a resulting SAS dataset named **new_data**, we deploy the following **PROC IMPORT** code block. The key components here are the mandatory use of `dbms=csv` to correctly interpret the comma as the field separator, and the inclusion of `getnames=YES`, which is critical for promoting the header row into legitimate SAS variable names.

```
/*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;
```

Upon successful execution of the code, the **PROC PRINT** statement is utilized to display the contents of the newly created dataset, **new_data**. This step serves as an immediate verification, allowing for a direct comparison with the original CSV file to confirm the integrity and accuracy of the data transfer.

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

The resulting output clearly demonstrates that the data was imported without error. The column headers were correctly interpreted as variable names, and all data values--both character and numeric--were precisely preserved and structured, confirming the seamless integration of the external CSV source into the SAS processing environment.

Example 2: Integrating Data from a Microsoft Excel Workbook

Microsoft Excel files are a ubiquitous format, particularly in corporate, research, and financial domains, prized for their ability to manage complex spreadsheets, handle multiple sheets, and support rich formatting. Importing these structured workbooks into SAS demands the specification of the appropriate **dbms** option to effectively navigate and interpret their complex, proprietary structure.

We will use an Excel file, **my_data.xlsx**, which contains the following tabular data located on its primary sheet. Unlike simple text files, Excel workbooks require SAS to initialize a specialized engine to read the cell-based data structure correctly.

	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						

To successfully import this spreadsheet data and create a SAS dataset named **new_data**, the following SAS code is executed. The critical parameter here is `dbms=xlsx`, which ensures that SAS correctly engages its built-in Excel engine. Since we are importing the primary sheet and assuming headers are present, `getnames=YES` remains standard.

```
/*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;
```

After the execution, the output from **PROC PRINT** validates the imported data. This confirmation step is essential in any data pipeline to verify that the transfer from the complex Excel structure to the standardized SAS format was flawless, particularly regarding data type assignment and

missing values.

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

The final SAS output perfectly replicates the original content of the Excel file. **PROC IMPORT** successfully identified the first row as variable names, preserved all data points, and ensured the resulting dataset is structured and immediately available for all subsequent analytical tasks within the SAS environment.

Example 3: Handling Custom Delimited Text Files (DLM)

Delimited text files (commonly found with `.txt` or `.dat` extensions) are often used when simple CSV or proprietary formats like Excel are unsuitable, or when custom field separation is required. These files rely on a specific, recurring character--known as a delimiter--to partition data values. When importing such files, the combination of `dbms=dlm` and the optional `DELIMITER=` specification is fundamental for accurate parsing.

Let us consider a text file named **data.txt**. In this example, the data is separated by a consistent delimiter (which, in the visual representation below, appears as spaces or tabs). The challenge here lies in instructing SAS to correctly interpret these separators as boundaries between fields.

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

To import this delimited text source into SAS and generate a dataset named **new_data**, we employ the following code. We select `dbms=dlm`, which signals to SAS that the file structure is generalized delimited text. In the absence of an explicit `DELIMITER=` option, SAS is often smart enough to infer common delimiters like tabs or multiple spaces. Crucially, `getnames=YES` ensures the first line is treated as the header row.

```
/*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 subsequent output generated by **PROC PRINT** confirms that the text file was accurately parsed according to the specified delimiter logic (whether inferred or explicit). This verification step

is vital, especially with DLM files, as inconsistent delimiters can easily lead to misaligned data columns.

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

A comparison between the SAS output and the original source confirms that all data values and their corresponding variable names were precisely extracted and structured within the SAS environment. This outcome underscores the adaptability of **PROC IMPORT** in handling custom delimited files, provided the user correctly identifies and manages the separator character.

Beyond the Core Syntax: Advanced `PROC IMPORT` Options

While the core syntax of **PROC IMPORT** is sufficient for routine tasks, realizing its full potential requires understanding specialized options designed to manage complex data scenarios, enhance control, and optimize performance. These advanced parameters are critical for creating robust, error-free data integration workflows, especially when dealing with large or non-standard external data file formats.

A common complexity arises when dealing with multi-sheet Excel files. The `SHEET=` option is essential here, allowing the user to specify precisely which worksheet should be imported (e.g., `SHEET="Sheet2"`). Similarly, for generic delimited files (where `dbms=dlm` is used), the `DELIMITER=` option is not just useful, but often mandatory for files using non-standard separators, such as defining `DELIMITER=";"` for semicolon-separated data or using hexadecimal notation like `DELIMITER='09'x` for tab-separated values.

Furthermore, SAS utilizes heuristic methods to guess variable types and lengths based on the initial rows of data. When data quality is inconsistent or the file is massive, these guesses can lead to incorrect data types or truncated character variables. The following advanced options allow for

granular control over this critical inference process:

DATAROW=: This option allows the user to explicitly define which row in the input file contains the absolute start of the data records. This is invaluable when the external file contains metadata, footers, or descriptive text preceding the actual dataset, ensuring SAS skips these non-data rows and begins reading from the correct starting line.
GUESSINGROWS=: By default, SAS samples a limited number of initial rows to determine the optimal data types and maximum lengths for variables. For files where the first few rows do not represent the full variability of the data, increasing the value of `GUESSINGROWS=` (or setting it to `GUESSINGROWS=MAX`) compels SAS to analyze more data. This significantly improves the accuracy of variable inferences, preventing issues like underestimated numeric precision or character string truncation.
SCANTEXT=: When SAS needs to determine the necessary storage length for character variables, it typically scans only a fraction of the data for efficiency. Using `SCANTEXT=YES` forces SAS to scan the entire input file for character variable lengths. While this might slightly increase processing time for extremely large files, it guarantees that no information is lost due to character truncation, providing maximum data integrity.

Employing these additional options provides the data analyst with the necessary fine-tuning capabilities to perfectly align the import process with the specific characteristics of complex external data files, thereby minimizing transformation errors and guaranteeing the integrity of the data destined for analysis.

Conclusion: Mastering Data Ingestion with PROC IMPORT

The **PROC IMPORT** statement is unequivocally an indispensable asset within the SAS programmer's arsenal. It provides a highly streamlined and efficient methodology for transferring disparate external data sources directly into the SAS analytical environment. By gaining proficiency in its core syntax and appreciating the immense versatility offered by the **dbms** argument, analysts are equipped to confidently handle a vast spectrum of file formats, ranging from simple CSVs and generic delimited text files to complex Microsoft Excel workbooks.

A successful, accurate data import is the foundational and most critical step in establishing any reliable data analysis workflow. Through the robust capabilities of **PROC IMPORT**, SAS empowers users to effectively mitigate common data integration challenges, thereby setting the stage for accurate statistical analysis, valid reporting, and insightful discoveries. We strongly recommend that users consistently consult the official SAS documentation for the most current and comprehensive details regarding all optional arguments, engine specifications, and best practices related to **PROC IMPORT**, ensuring full utilization of its capabilities.

Note: The SAS Documentation for PROC IMPORT provides a complete and exhaustive reference

detailing all optional arguments available for file import, including specific engine options tailored for various file types and database connections.

Additional Resources

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