How to Use the INFILE Statement in SAS (With Example)

Authored by

stats writer

November 19, 2025

The INFILE statement is a fundamental component of the SAS Data Step, employed specifically for importing external data sources into a structured SAS dataset. This powerful statement facilitates the reading of various file types, including plain text files, raw data streams, and files formatted as CSV or other delimited formats. It is essential for defining the physical location of the input file, specifying the data's structure, and controlling various input processing attributes and options.

Utilizing the INFILE statement allows the user to precisely manage how raw data records are processed and interpreted by the SAS system. It is always followed by a path to the external file and a sequence of carefully selected options that dictate crucial aspects of the reading process, such as handling delimiters, managing missing values, and defining record lengths. For instance, the statement `INFILE 'filename.csv' DELIMITER=',' MISSOVER DSD LRECL=200;` instructs SAS to read a comma-separated file, permitting missing values (MISSOVER), treating consecutive delimiters as missing (DSD), and setting the maximum record length (LRECL) to 200 characters.

Mastering the options available within the INFILE statement is key to successful data integration and preparation within the SAS environment. The subsequent sections will provide a detailed breakdown of its structure and practical applications.

Understanding the Basic INFILE Syntax

The primary use case for the **INFILE** statement is facilitating the efficient transfer of external data into a temporary or permanent SAS dataset. This process is executed within a Data Step, which is the foundational block for data manipulation in SAS.

The standard structure begins with the DATA statement, followed immediately by the **INFILE** statement, which specifies the data source and necessary parsing options. Finally, the INPUT statement defines how the raw data fields map to SAS variables. This overall structure is demonstrated below:

```
data my_data;
infile '/home/u13181/bball_data.txt' dlm=' ' dsd missover firstobs=2;
input team $ position $ points assists;
run;
```

Each line in this basic Data Step serves a distinct and crucial function in the import process:

data: Specifies the name assigned to the new SAS dataset created during the Data Step execution (e.g., `my_data`).
infile: Indicates the exact file path and name of the external source file containing the raw data records that SAS must process.

dlm: Defines the delimiter character(s) that separate the data values within the input file (e.g., a space, comma, or tab).
dsd: Stands for Delimiter-Sensitive Data. When used, it instructs SAS to treat consecutive delimiters as representing a missing value rather than ignoring them, which is critical for maintaining data structure alignment.
missover: Ensures that if SAS reaches the end of a physical record without finding values for all specified variables in the INPUT statement, the remaining variables are assigned missing values instead of reading data from the next physical record.
firstobs: Specifies the line number in the input file where SAS should begin reading the data observations, effectively skipping header rows or introductory metadata.
input: Defines the names, types (character indicated by $), and order of the variables to be read from the input file records.

Detailed Explanation of Key INFILE Options

While the basic syntax provides the framework, the true flexibility of the INFILE statement lies in the multitude of options available to handle diverse external file formats and complexities. Proper selection of these options is paramount for accurate data parsing, especially when dealing with inconsistencies such as quoted strings, embedded delimiters, or variable-length records.

Options like **DLM=** and **DSD** are essential when working with delimited files. The **DLM=** option explicitly tells SAS which character separates the fields. If the file is a standard CSV, `DLM=','` is used. However, the presence of **DSD** changes how SAS processes these delimiters. Without **DSD**, SAS treats multiple contiguous delimiters as one, which can cause data misalignment if a field truly holds a missing value between two delimiters. Using **DSD** corrects this by forcing SAS to interpret the gap as a missing value and also strips quotation marks from character variables, simplifying the handling of text fields containing internal delimiters.

Furthermore, options controlling record management, such as **MISSOVER** and **TRUNCOVER**, are vital. **MISSOVER** is generally preferred when records might be shorter than expected, ensuring that SAS assigns missing values to variables that run past the end of the line, thereby preventing the program from reading the next physical record prematurely. Conversely, **TRUNCOVER** allows SAS to truncate data values if the field length exceeds the specified format length, providing another method to handle irregularities in record structure efficiently.

Practical Example: How to Use INFILE Statement in SAS

To illustrate the practical application of the Data Step structure and the associated INFILE options, we will use a sample file containing basketball statistics. This example highlights how to handle common data challenges, specifically header rows and internal missing values, during the import

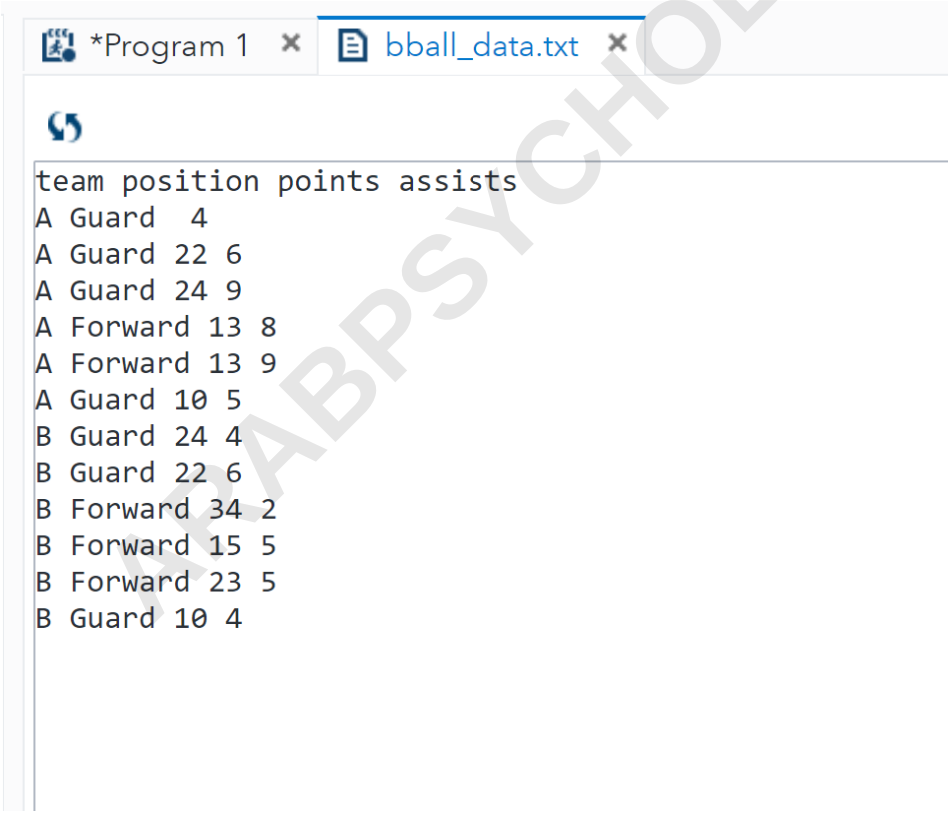
process.

The objective is to import the data from a simple text file into a SAS dataset named `my_data`, ensuring that the descriptive header row is skipped and that empty fields are correctly recognized as missing values.

Analyzing the Sample Data File

Consider the structure of the external text file, designated **bball_data.txt**, which contains team, player position, points, and assists data. Examination of this file reveals two key structural features that must be addressed during the SAS import process: a descriptive header row and variability in data completeness.

Specifically, the first line contains variable descriptions (Team, Position, Points, Assists) which should not be imported as data observations. Furthermore, note the first data row where the value for "Points" is clearly missing, indicating the need for careful delimiter and missing value handling within the INFILE statement to maintain alignment.



Since the fields in this file are separated by spaces, we will utilize `DLM=' '`. To ensure the missing value in the third column of the first observation is correctly identified, the **DSD** option must be employed alongside **MISSOVER**.

Implementing the SAS Code

The following SAS code block demonstrates the complete Data Step required to import the `bball_data.txt` file. Note how the INFILE options are used in conjunction with the INPUT statement to define the variable attributes and structure the data correctly.

```
/*import data from txt file into SAS dataset*/
data my_data;
infile '/home/u13181/bball_data.txt' dlm=' ' dsd missover firstobs=2;
input team $ position $ points assists;
run;

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

Upon execution of this code, the `proc print` statement displays the resulting SAS dataset, `my_data`. The output confirms that the raw textual data has been accurately transformed into a structured table, respecting the specified delimiters and handling the missing value as intended.

Obs	team	position	points	assists
1	A	Guard	.	4
2	A	Guard	22	6
3	A	Guard	24	9
4	A	Forward	13	8
5	A	Forward	13	9
6	A	Guard	10	5
7	B	Guard	24	4
8	B	Guard	22	6
9	B	Forward	34	2
10	B	Forward	15	5
11	B	Forward	23	5
12	B	Guard	10	4

The resulting dataset, as shown in the output above, demonstrates that the first record now correctly displays a missing value for the `Points` variable, and the header row was successfully omitted from the dataset creation process.

Interpreting the Results and Options

The successful execution of the Data Step confirms the power and necessity of the **INFILE** statement in managing external data importation. By meticulously defining the file path and parsing rules, we achieved a clean dataset structure despite the initial complexities present in the raw data file.

A closer look at the specific arguments used reveals how they addressed the file's inherent structure:

INFILE: Required to provide the absolute path `'/home/u13181/bball_data.txt'`, ensuring SAS could locate the necessary input source for processing.
DLM=' ': Critically identified the space character as the primary delimiter, allowing SAS to segment the data fields accurately.
DSD: This option was essential for handling data integrity, specifically by treating the sequence of two consecutive space delimiters between the `Position` and `Points` fields in the first record as a legitimate missing numeric value, rather than shifting subsequent fields leftward.
MISSOVER: Ensured that if a record ended early (which was not the case here, but is standard practice), SAS would assign missing values to unread variables instead of attempting to read beyond the current line, preserving the one-to-one record-to-observation mapping.
FIRSTOBS=2: Directly addressed the presence of the header row by instructing SAS to commence reading observations starting from the second physical line of the input file.
INPUT: Defined the variable structure (`team $ position $ points assists`), assigning appropriate names and data types (character '$' or numeric) to the resulting columns in the SAS dataset.

The combined effect of these carefully selected arguments demonstrates how the INFILE statement provides granular control over the crucial process of transforming unstructured data into a standardized, analytical SAS format. This methodology is applicable across diverse external data sources, from basic text files to complex CSV dumps.

Conclusion and Summary

The INFILE statement is indispensable for any SAS programmer needing to integrate external data into their analytical environment. It offers precise control over input file characteristics, ranging from simple delimiter specification to complex handling of missing data and record lengths. By understanding and applying options like DLM, DSD, MISSOVER, and FIRSTOBS, users can reliably convert raw text files into clean, ready-to-analyze SAS datasets.

Successful data importation often hinges on careful inspection of the source file to determine the correct set of options required by the INFILE statement. This attention to detail ensures that the

resulting dataset accurately reflects the information in the original source, setting the stage for robust statistical analysis.

For those looking to explore more advanced methods of data input or different file types, the following resource provides further guidance on text file importation:

How to Import Text Files into SAS