A Comprehensive Guide to Importing Data into SAS using the INFILE Statement

Authored by

Mohammed looti

November 14, 2025

In the expansive realm of data analysis and statistical programming, the ability to efficiently ingest external data is not just important--it is foundational. For professionals utilizing **SAS** (Statistical Analysis System), the **INFILE statement** stands out as the primary, highly versatile utility for reading raw data from external sources, such as plain **text files** or comma-separated value (CSV) files, and transforming them into a structured **SAS dataset**. This capability is absolutely essential for researchers, analysts, and data scientists who must frequently integrate data collected outside of the native SAS environment for subsequent reporting and modeling.

The inherent flexibility of the **INFILE statement** is what makes it indispensable. It empowers users to tackle highly diverse file formats and complex structural arrangements within raw data, thereby enabling crucial data preparation steps. By meticulously specifying the file's location and defining a set of rules for how **SAS** should interpret its contents--be it delimited, fixed-column, or list input--you effectively bridge the gap between unstructured raw data and the highly structured environment of a SAS library. This article will provide a comprehensive exploration of the core syntax and practical application of the **INFILE statement**, illustrated through a clear, step-by-step example.

The Crucial Role of the INFILE Statement in SAS Data Steps

The fundamental purpose of the **INFILE statement** is to direct **SAS** on exactly how and where to read raw data during the execution of a **DATA step**. It serves as the initial declaration, establishing the link between your source data file (which exists outside the SAS system) and the new **SAS dataset** being created. Without this critical instruction, the SAS program would halt, unable to locate or interpret the necessary input data structure.

In standard practice, the **INFILE statement** is declared first, specifying the complete file path, followed by a series of specialized **options** that precisely control the reading mechanism. These options are vital for managing data nuances such as defining the **delimiter**, handling missing records, and excluding extraneous lines like header information. The power of data ingestion is fully realized when the INFILE statement is correctly paired with the **INPUT statement**, which defines the names, data types, and order of the incoming **variables**.

To visualize the fundamental syntax for importing data from a standard delimited **text file**, consider the following SAS code structure. Notice how the DATA step initiates the creation of the dataset, the INFILE command points to the physical file and sets the reading rules, and the INPUT command specifies the structure of the data records being read:

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

This concise code block demonstrates the core process of reading external data. While the DATA and RUN statements frame the operation, the synergistic relationship between the **INFILE statement** (source and rules) and the **INPUT statement** (structure and variable definition) is what ultimately translates raw lines of text into the columns and rows of a usable **SAS dataset**. We will examine the specific options used here to clarify their critical functions.

Essential INFILE Options for Robust Data Handling

The true power of the INFILE statement is unlocked by its wide array of specialized **options**, which provide granular control over the data parsing and import process. Mastering these configurations is crucial for successfully loading data regardless of its external format--whether it is perfectly structured or contains errors and inconsistencies. Below is a detailed breakdown of the most frequently used options and their roles in data ingestion:

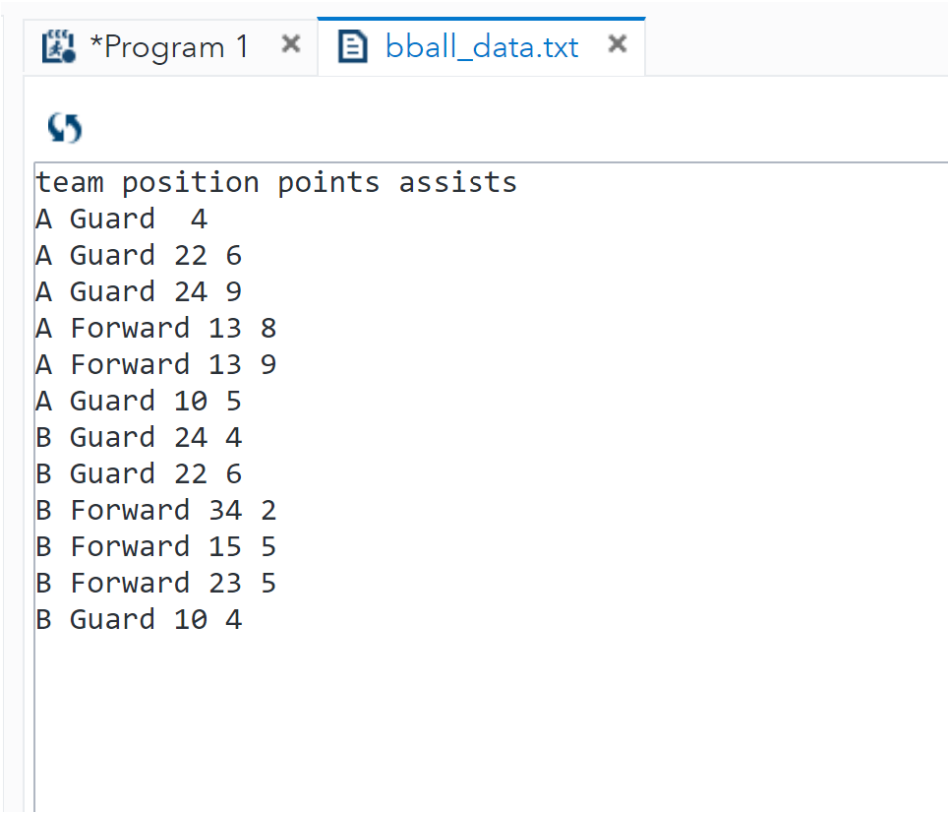
DATA Statement: This command, appearing at the beginning of the **DATA step**, is not an INFILE option but is fundamental to the process, as it names the resultant **SAS dataset** that will be created upon successful import.
INFILE Path: This argument specifies the exact, complete path and filename of the external **text file** from which **SAS** will read the raw data lines.
DLM (Delimiter): This key **option** defines the specific character used to separate data **values** (fields) within each record. Commonly used **delimiters** include spaces, commas (for CSV), tabs, or vertical bars.
DSD (Delimiter-Sensitive Data): When applied, DSD modifies the standard behavior of list input. Critically, it instructs SAS to interpret two consecutive **delimiters** as a missing value, which is essential when dealing with standard comma-separated files where missing fields are common. Furthermore, DSD enables the input of quoted strings that might contain the delimiter itself.
MISSOVER: This **option** manages incomplete data records. It prevents the **INPUT statement** from crossing into the next line if the current data record ends prematurely (i.e., if it has fewer data fields than expected). Instead, it assigns system missing **values** to any remaining variables specified in the INPUT statement, ensuring that one physical line corresponds to exactly one logical **observation**.
FIRSTOBS: This powerful **option** dictates the line number at which SAS should commence reading actual data. It is most commonly set to `FIRSTOBS=2` to effectively skip a single header row in the input file that typically contains variable names.

The careful configuration of these options, coupled with the proper sequencing and typing specified in the **INPUT statement** (where a dollar sign **$** denotes a character variable), ensures that external raw data is accurately ingested and prepared for subsequent statistical processing within the SAS environment.

Practical Application: Importing a Space-Delimited File

To solidify our understanding of the INFILE statement, let us walk through a typical import scenario: loading basketball player statistics from a space-delimited raw **text file**. We assume the source file, named **bball_data.txt**, resides at the path **'/home/u13181/bball_data.txt'**. This file contains four key fields: team, position, points scored, and assists.

Data inspection is the crucial first step. We must examine the raw file structure to accurately determine the **delimiter**, identify any header lines that must be skipped, and anticipate how missing **values** are represented. Observing the content of **bball_data.txt** reveals the following structure:



As clearly depicted in the image, the input file contains a header row listing the field names ("team", "position", "points", and "assists")--meaning we must start reading actual data from line two. The data fields are separated by a single space, establishing the space character as our required delimiter. Furthermore, we can spot a common data imperfection: the first data entry is missing a value in the "points" column. This pattern of missing data necessitates the use of specific INFILE options to ensure proper handling and prevent data misalignment.

Based on these findings, our objective is to construct a **SAS** program that correctly skips the header, utilizes the space delimiter, and correctly assigns a system missing value where data is

absent, thus accurately transforming the raw file into a structured **SAS dataset**.

Implementing and Verifying the SAS Import Code

Having analyzed the source file structure, we can now write the SAS program to import the data into a new **SAS dataset** named **my_data**. The code snippet below integrates the necessary **INFILE options** (DLM=' ', DSD, MISSOVER, and FIRSTOBS=2) specifically chosen to match the characteristics of our **bball_data.txt** file:

```
/*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, this code successfully reads the raw data, and the contents of **bball_data.txt** are converted into the tabular structure of a SAS dataset. We use the **PROC PRINT** statement immediately afterward to display and confirm the integrity of the newly created data structure. The resulting output demonstrates the success of the import process:

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

As confirmed by the output, the raw text has been perfectly structured into the **my_data SAS**

dataset. The header row was correctly bypassed, and most importantly, the missing value in the "points" column for the first record was accurately represented by a period (.), which is the standard symbol for a missing numeric **variable** in **SAS**.

Detailed Analysis of the INFILE Arguments

Understanding why each argument was selected is crucial for replicating successful data imports. Let's break down the specific components of the INFILE statement used in the basketball data example:

INFILE Path Argument: `'/home/u13181/bball_data.txt'`. This provided the essential instruction, giving SAS the absolute path to locate and open the external **text file**, initiating the read process.
DLM Option: `=' '`. By setting the **delimiter** to a single space, we explicitly told SAS how to parse the fields, ensuring that data elements like "Mavs", "Guard", and "25" were correctly recognized as distinct **values**.
DSD Option: This option was indispensable here. Since the missing point score resulted in two implied spaces functioning as consecutive delimiters, DSD ensured that SAS correctly interpreted the gap as a missing **value** for the numeric 'points' variable, rather than incorrectly shifting subsequent data elements.
MISSOVER Option: This safeguard ensured that if any record was unexpectedly short (i.e., missing trailing values), the **INPUT statement** stopped reading on the current line, assigning missing values to the remaining **variables** and preventing the program from wrapping to the next line and corrupting the subsequent **observation**.
FIRSTOBS Option: `=2`. This crucial setting achieved the primary goal of skipping the non-data header line. By starting the read operation on line two, we ensured that the variable names were ignored, and only the actual player data was processed.
INPUT Statement: `team $ position $ points assists;`. This defined the four columns in the output dataset. The inclusion of the dollar sign (**$**) correctly identified `team` and `position` as character variables, while `points` and `assists` defaulted to numeric types.

The successful combination of these arguments illustrates how the INFILE statement provides the precision necessary to handle real-world raw data, transforming unstructured files into analytical assets within the SAS environment.

Conclusion: Best Practices for Mastering Data Import

The **INFILE statement** is undeniably a cornerstone of effective data management in **SAS**, offering robust functionality for importing raw data from various external **flat files**. Its capacity to handle complex data structures through powerful **options** allows analysts to precisely control the

transformation of raw input into an actionable **SAS dataset**.

To ensure consistent and error-free data ingestion, data professionals should always adhere to the following best practices: **First**, perform a thorough initial inspection of the raw data file; understand its **delimiter**, encoding, and the presence of any non-data lines like headers. **Second**, select and meticulously configure the necessary **INFILE options** (such as DSD for missing data or FIRSTOBS for skipping headers) that perfectly align with your file's specific characteristics. **Third**, always verify the integrity of the imported **SAS dataset** by using inspection procedures like **PROC PRINT** to examine the observations, or **PROC CONTENTS** to confirm correct **variable** names and data types.

Proficiency in the **INFILE statement** is a hallmark of skilled SAS programming, enabling the efficient preparation of diverse external data for sophisticated statistical analysis and predictive modeling.

Additional Resources

How to Import Text Files into SAS