# Learning Data Embedding in SAS: A Practical Guide to the CARDS Statement

Authored by

**Mohammed looti**

November 14, 2025

The SAS CARDS statement represents a fundamental and efficient approach to embedding raw data directly within your program source code. This intrinsic technique allows programmers to seamlessly input observational values into a new dataset without the complexity of managing external files, defining file paths, or executing elaborate import procedures. This capability proves exceptionally valuable when working with smaller data subsets, conducting rapid prototyping or testing scenarios, or when the goal is to create self-contained, easily reproducible code examples for documentation or sharing.

For any serious SAS programmer, mastering the effective application of the CARDS statement is essential, as it significantly streamlines data management processes directly within the core logic of the data step. This comprehensive guide is designed to provide precise instruction, detailing the required syntax, illustrating its practical use through coding examples, and clarifying its relationship with the functionally identical, modern alternative: the DATALINES statement.

## Defining the Core CARDS Statement Syntax

Integrating the CARDS statement into a SAS program mandates a highly specific and structured sequence of instructions, all contained within the data step. This systematic structure is crucial because it communicates three critical pieces of information to the SAS execution engine: first, the designated name of the new dataset being generated; second, the precise names and corresponding data types of all variables it will contain; and finally, the exact demarcation points--where the raw data values commence and where they conclude.

The fundamental syntax for defining and populating a dataset using this in-stream method requires the definition of the program logic to precede the raw input data immediately. A critical rule governing this method is the termination of the raw data block: it must be concluded by a single semicolon (`;`) placed entirely on its own line. This solitary semicolon acts as the definitive end-of-file marker for the data stream, allowing SAS to correctly parse the subsequent program statements.

To demonstrate this structure, the following example illustrates the most basic code block required to initialize and populate a simple dataset named `my_data` using the in-stream input methodology. This concise code defines two variables, inputs the values, and signals the end of the data step:

```
data my_data;
input var1 $ var2;
cards;
A 12
B 19
C 23
D 40
```

```
;
run;
```

This succinct block of code successfully executes a complete data step, resulting in the creation of a temporary or permanent dataset. Specifically, the dataset `my_data` is constructed with two variables: `var1`, which is designated as a character variable (indicated by the crucial dollar sign, $), and `var2`, which is automatically treated as numeric. The data itself is sourced immediately following the CARDS statement and processing ceases precisely upon encountering the required terminating semicolon.

## Deconstructing Key Data Step Components

To proficiently utilize in-stream data entry techniques, it is necessary to move beyond the high-level concept and understand the distinct, functional role played by each command within the sequential flow. Each statement contributes uniquely to the successful compilation and execution of the data step, collectively defining everything from the dataset's identity to the internal variable structure and the data source location.

The following list systematically breaks down the function and purpose of the four core statements that are invariably used in conjunction with the `CARDS` or `DATALINES` statements:

**`data` Statement:** This command signals the absolute beginning of a data step block in SAS. It must be immediately followed by the programmer-chosen name for the resulting dataset. For example, `data Sales_Report;` instructs SAS to prepare the environment to create a dataset named `Sales_Report`.
**`input` Statement:** The `input` statement is responsible for the crucial task of mapping the raw data values to meaningful fields. Here, you define the names of all variables and their data types, ensuring that the defined order precisely matches the sequence of values provided in the data lines that follow.
**`cards` Statement:** As the main focus, the CARDS statement acts as a clear, mandatory delimiter. It explicitly notifies SAS that the subsequent lines contain the actual, raw data values corresponding to the variables defined in the `input` statement, separating the program logic from the observational data.
**`run` Statement:** The `run` statement is essential as it signals the successful conclusion of the current data step block. Upon encountering this statement, SAS executes the compiled instructions, processes the in-stream data, and finalizes the creation of the specified dataset.

A fundamental concept to grasp when defining your variables within the `input` statement is the explicit type declaration using the dollar sign (`$`). Placing this symbol directly after a variable name (e.g., `EmployeeName $`) explicitly instructs SAS to interpret and store this field as a character

variable, which is capable of holding alphanumeric text strings. Conversely, if the dollar sign is omitted, the SAS system defaults to assuming the variable holds standard numeric values. Correct type definition is absolutely critical for accurate data interpretation, storage, and subsequent statistical analysis.

## The Historical Context and Legacy of CARDS

The somewhat antiquated nomenclature of the CARDS statement is not arbitrary; it provides a fascinating historical window into the early decades of computer science and legacy data input methodology. The term "CARDS" is a direct, enduring reference to the era when both program instructions and observational data were physically handled and processed using stacks of punch cards.

Prior to the development and widespread deployment of modern magnetic storage media and interactive terminal input systems, programmers relied heavily on Hollerith cards (or punch cards) to feed sequential data into massive mainframe computers. In this system, every line of programming code or data was represented by a physical card, with information encoded by the presence or absence of holes punched at specific grid locations. Critically, after the program instructions were successfully loaded, the physical stack of corresponding raw data cards was fed in immediately afterward.

Consequently, when a programmer employs the CARDS statement in modern SAS programming, it serves as a metaphorical, digital placeholder for that historical physical stack of data cards that would traditionally follow the code block. This powerful legacy naming convention remains a valuable, though subtle, reminder of the evolution of programming languages and the foundational methods of data entry that characterized the early days of computing, even though the data is now input digitally in-stream.

## Practical Application: Creating a Sports Statistics Dataset

To gain a full appreciation for the convenience and efficiency offered by the CARDS statement, we will walk through a detailed, practical example focused on sports statistics. Our objective is to generate a SAS dataset that records team performance metrics, requiring the definition of three distinct variables: `team` (declared as a character variable), and both `points` and `assists` (which will be numeric variables).

The following SAS code block illustrates the complete, functional flow, starting with the initialization of the data step, proceeding to the definition of the variables via the `input` statement, introducing the raw data using the `CARDS` delimiter, and finally, executing the routine with the `run` statement. This entire block is a self-contained unit ready for execution:

```
/*create dataset*/
data my_data;
input team $ points assists;
cards;
Mavs 14 9
Spurs 23 10
Rockets 38 6
Suns 19 4
Kings 30 4
Blazers 19 6
Lakers 22 14
Heat 19 5
Magic 14 8
Nets 27 8
;
run;
/*view dataset*/
proc print data=original_data;
```

| Obs | team | points | assists |
|---|---|---|---|
| 1 | Mavs | 14 | 9 |
| 2 | Spurs | 23 | 10 |
| 3 | Rockets | 38 | 6 |
| 4 | Suns | 19 | 4 |
| 5 | Kings | 30 | 4 |
| 6 | Blazers | 19 | 6 |
| 7 | Lakers | 22 | 14 |
| 8 | Heat | 19 | 5 |
| 9 | Magic | 14 | 8 |
| 10 | Nets | 27 | 8 |

Upon successful execution, SAS compiles and creates the dataset named `my_data`, accurately reflecting the ten rows of observational data provided in-stream. The system correctly assigns team names as character values and the points and assists as numeric values, precisely following the definitions established within the `input` statement. It is important to note that the PROC PRINT statement is generally used immediately after this process to display the contents of the newly

created data set (and should reference `my_data`, although the illustration uses `original_data` as a generic placeholder).

## Exploring the Modern Alternative: The DATALINES Statement

While the historically rooted CARDS statement remains fully supported across all modern SAS versions, its contemporary alternative, the DATALINES statement, is increasingly favored in current SAS programming environments. The primary reason for this preference is purely stylistic: `DATALINES` is functionally more descriptive and intuitive regarding its true purpose--marking the beginning of the data lines.

A crucial operational fact for programmers to remember is that, despite the difference in naming, the CARDS statement and the DATALINES statement are completely and perfectly interchangeable; they execute the identical command within the data step execution sequence. Both statements serve the singular function of explicitly signaling that the lines immediately following contain the raw input data intended to be read into the dataset currently under construction.

To demonstrate their functional equivalence, we can modify the previous sports statistics example simply by replacing the `CARDS` keyword with `DATALINES`. The result, as shown below, is the creation of an identical dataset, proving that the choice between the two is ultimately a matter of programmer preference and coding standards adherence:

```
/*create dataset*/
data my_data;
input team $ points assists;
datalines;
Mavs 14 9
Spurs 23 10
Rockets 38 6
Suns 19 4
Kings 30 4
Blazers 19 6
Lakers 22 14
Heat 19 5
Magic 14 8
Nets 27 8
;
run;
/*view dataset*/
```

**proc print data=original_data;**

| Obs | team | points | assists |
|---|---|---|---|
| 1 | Mavs | 14 | 9 |
| 2 | Spurs | 23 | 10 |
| 3 | Rockets | 38 | 6 |
| 4 | Suns | 19 | 4 |
| 5 | Kings | 30 | 4 |
| 6 | Blazers | 19 | 6 |
| 7 | Lakers | 22 | 14 |
| 8 | Heat | 19 | 5 |
| 9 | Magic | 14 | 8 |
| 10 | Nets | 27 | 8 |

As the structural output confirms, the dataset generated through the use of `DATALINES` is structurally and numerically equivalent to the one produced using the `CARDS` statement. While SAS maintains full support for `CARDS` to ensure backward compatibility and ease of use, many contemporary programmers opt for the enhanced clarity afforded by the DATALINES statement, particularly when initiating new projects or establishing team coding standards.

## Conclusion: Mastering In-Stream Data Input in SAS

The CARDS statement, alongside its modern, functional synonym, the DATALINES statement, provides an indispensable and efficient mechanism for embedding small-scale data payloads directly within the source code of your SAS programs. This methodology remains highly effective for scenarios requiring rapid prototyping, generating clear instructional examples, and managing contained data tasks without incurring the complexity or overhead associated with configuring and managing external data files.

Achieving success in data manipulation relies on a strong, fundamental comprehension of the core syntax of the data step. Specifically, programmers must ensure the proper sequential ordering of the `data`, `input`, and the `cards`/`datalines` statements. Crucially, always verify that your in-stream data block is precisely and definitively terminated using a semicolon (`;`) placed alone on its own line, and that the entire data step concludes with a final `run` statement to trigger the execution phase and the subsequent dataset creation.

While the majority of large-scale, enterprise-level data analyses necessarily depend on robust

procedures for importing data from various external databases and files, the capacity to quickly define and load data in-stream is an invaluable skill. This technique dramatically enhances the efficiency, clarity, and overall readability of your SAS programming toolkit, making it ideal for diagnostics and quick demonstration tasks.

## Further Resources for Advanced SAS Programming

To facilitate the continued expansion of your knowledge base and advanced skills in SAS programming and data analysis, we highly recommend exploring the following authoritative and trustworthy resources that provide in-depth tutorials, comprehensive documentation, and insights into advanced topics:

**SAS Official Documentation:** Access comprehensive guides detailing all SAS statements, procedures, and functional components.
**SAS Global Forum Papers:** Review academic and industry papers that offer advanced insights and real-world applications from experts across various fields.
**Online SAS Communities:** Engage with the wider community of SAS users to troubleshoot specific issues, share best practices, and collaborate on complex programming challenges.