Learning to Use FIRST. and LAST. Variables for Group Processing in SAS

Authored by

Mohammed looti

October 30, 2025

In the complex environment of data manipulation and analytical programming, particularly within the SAS system, the ability to effectively manage and summarize grouped data is paramount. Many critical tasks--from calculating subtotals to extracting unique entries--require precise identification of the boundaries of these groups. This is where the powerful implicit features of **FIRST.** and **LAST.** variables come into play. These variables are binary flags that allow programmers to precisely target the initial and final observation within distinct sets of records, enabling highly controlled data processing logic.

This comprehensive guide is designed to clarify the utility, mechanics, and practical application of the **FIRST.** and **LAST.** variables in SAS programming. We will detail how these temporary variables are generated, examine their role in conditional processing, and demonstrate how they can be leveraged to streamline group-wise operations on your dataset. Mastering these concepts is fundamental for any professional seeking to write concise and efficient SAS code for data management and analytical workflows.

Understanding the Mechanics of FIRST. and LAST. Variables

The implicit **FIRST.** and **LAST.** variables are automatically created by SAS whenever a DATA step includes a BY statement. These are temporary, non-output variables that serve as crucial logical flags, indicating whether the currently processed observation represents the beginning or the end of a group defined by the specified BY variables. This mechanism provides the context necessary for loop-based processing within the DATA step.

To properly initialize and utilize these variables, a critical prerequisite must be met: your dataset must be sorted by the variable(s) listed in the BY statement. This sorting ensures that all observations belonging to the same group are physically contiguous within the data file, allowing SAS to correctly recognize group transitions during sequential processing. The PROC SORT procedure is almost always necessary immediately before the DATA step that employs these implicit variables.

The functional behavior of these variables is binary, making them ideal for use in conditional logic (e.g., IF-THEN statements):

FIRST.variable_name: This variable evaluates to **1** (True) only for the initial observation encountered within the group defined by **variable_name**. For all subsequent records within that group, its value is **0** (False). This flag is typically used to initialize counters, assign group starting values, or extract header records.
LAST.variable_name: Conversely, this variable evaluates to **1** (True) only for the final observation encountered in the group defined by **variable_name**. All other records in the group hold a value of **0** (False). This flag is essential for finalizing cumulative calculations, writing summary statistics, or identifying trailer records.

Establishing the Example Dataset for Group Processing

To effectively demonstrate how the **FIRST.** and **LAST.** variables function, we will utilize a practical example involving basketball team performance data. This simple dataset, named `my_data`, includes multiple records for three distinct teams, tracking their individual points scored and rebounds. This structure perfectly mimics real-world data where grouping and summarizing are necessary.

The following DATA step code is used to construct and populate this example dataset. Following its creation, we employ PROC PRINT to visualize the raw, unsorted data, providing a baseline understanding of the data structure before we apply any grouping logic.

```
/*create dataset*/
data my_data;
input team $ points rebounds;
datalines;
Mavs 29 10
Mavs 13 6
Mavs 22 5
Mavs 20 9
Spurs 13 9
Spurs 15 10
Spurs 33 8
Spurs 27 11
Rockets 25 8
Rockets 14 4
Rockets 16 7
Rockets 12 4
;
run;

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

Obs	team	points	rebounds
1	Mavs	29	10
2	Mavs	13	6
3	Mavs	22	5
4	Mavs	20	9
5	Spurs	13	9
6	Spurs	15	10
7	Spurs	33	8
8	Spurs	27	11
9	Rockets	25	8
10	Rockets	14	4
11	Rockets	16	7
12	Rockets	12	4

The output confirms that the `my_data` contains three clearly distinct groups: Mavs, Spurs, and Rockets. Our primary objective in the following sections is to utilize the implicit **FIRST.** and **LAST.** variables, coupled with the BY statement, to correctly identify and target the initial and final records associated with each team.

Leveraging FIRST. Variables to Identify Group Starts

The **FIRST.** variable is exceptionally valuable for executing operations that must only occur once at the start of a new group. These operations typically include initializing running totals, resetting iteration counters, or creating a unique identifier that applies to all records in the group. However, before the **FIRST.** variable can be utilized, the data must be correctly ordered.

The initial step must always be to sort the dataset by the grouping variable using PROC SORT. This ensures that when the DATA step iterates through the records, the **FIRST.** flag accurately identifies the precise moment the group changes. We will now demonstrate how to create a new flag variable that explicitly labels the starting record for each team in our basketball dataset.

Practical Application: Identifying First Observations

To implement the **FIRST.** variable correctly, we must first sort `my_data` by the `team` variable using PROC SORT. Subsequently, within the DATA step, we use the BY statement to group the records and then assign the value of `first.team` to a new variable, `first_team`.

/*sort dataset by team*/

```
proc sort data=my_data;
by team;
run;

/*create new dataset that labels first row for each team*/
data first_team;
set my_data;
by team;
first_team=first.team;
run;

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

Obs	team	points	rebounds	first_team
1	Mavs	29	10	1
2	Mavs	13	6	0
3	Mavs	22	5	0
4	Mavs	20	9	0
5	Rockets	25	8	1
6	Rockets	14	4	0
7	Rockets	16	7	0
8	Rockets	12	4	0
9	Spurs	13	9	1
10	Spurs	15	10	0
11	Spurs	33	8	0
12	Spurs	27	11	0

The resulting `first_team` dataset clearly shows the new `first_team` variable. Notice that this variable is set to **1** for the initial observation of Mavs, Rockets, and Spurs, and reverts to **0** for all subsequent records within those groups. This confirms the efficacy of the **FIRST.** variable in distinguishing the start of each group boundary.

Beyond simple flagging, the **FIRST.** variable is commonly used as a powerful conditional filter. By using the shorthand `if first.team;` within the DATA step, we can instruct SAS to retain only the first record for every group. This is an indispensable technique for tasks requiring data deduplication or the extraction of unique identification records.

```
/*sort dataset by team*/
proc sort data=my_data;
by team;
run;

/*create new dataset only contains first row for each team*/
data first_team_filtered;
set my_data;
by team;
if first.team;
run;

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

Obs	team	points	rebounds
1	Mavs	29	10
2	Rockets	25	8
3	Spurs	13	9

As expected, the `first_team_filtered` dataset now contains only a single observation for each team. The simplicity and efficiency of the `if first.team;` logic demonstrate its power: since `first.team` is treated as a Boolean condition, SAS only writes an observation to the output file when the flag is true (1).

Utilizing LAST. Variables to Conclude Group Operations

Complementing the `FIRST.` variable, the **LAST.** variable signals the definitive end of a processing group. This is crucial for tasks that require finalization actions, such as outputting summary records, calculating overall group averages, or generating end-of-group reports. Like its counterpart, the effectiveness of the **LAST.** variable hinges entirely on the prerequisite sorting of the data by the relevant grouping variable.

In this next demonstration, we will follow the same methodical approach: first ensuring the data is sorted by team using PROC SORT. We will then execute a DATA step that creates a new binary variable, `last_team`, which is assigned the value of `last.team`. This clearly illustrates which observation marks the conclusion of each team's set of records.

Practical Application: Identifying Last Observations

To properly isolate the final record per team using the **LAST.** variable, the data must first be ordered by `team` via PROC SORT. Once the sorted order is established, the subsequent DATA step, which includes the necessary BY statement, will correctly populate the `last_team` variable.

```
/*sort dataset by team*/
proc sort data=my_data;
by team;
run;

/*create new dataset that labels last row for each team*/
data last_team;
set my_data;
by team;
last_team=last.team;
run;

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

Obs	team	points	rebounds	last_team
1	Mavs	29	10	0
2	Mavs	13	6	0
3	Mavs	22	5	0
4	Mavs	20	9	1
5	Rockets	25	8	0
6	Rockets	14	4	0
7	Rockets	16	7	0
8	Rockets	12	4	1
9	Spurs	13	9	0
10	Spurs	15	10	0
11	Spurs	33	8	0
12	Spurs	27	11	1

Reviewing the `last_team` output, we observe that the `last_team` variable holds a value of **1** only for the final record of Mavs, Rockets, and Spurs. This demonstrates how the **LAST.** variable

effectively flags the group conclusion, enabling developers to perform conditional actions specifically at the end of a group.

Just like the `FIRST.` variable, the **LAST.** variable can be used as a conditional filter to create a summary dataset containing only the concluding record for each group. This technique is invaluable when tracking cumulative metrics or when the most recent entry per category is needed for downstream analysis.

```
/*sort dataset by team*/
proc sort data=my_data;
by team;
run;

/*create new dataset only contains last row for each team*/
data last_team_filtered;
set my_data;
by team;
if last.team;
run;

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

Obs	team	points	rebounds
1	Mavs	20	9
2	Rockets	12	4
3	Spurs	27	11

The filtered output, `last_team_filtered`, contains only the final observation for each team. The simplicity of the `if last.team;` statement efficiently handles the filtering process, writing a record only when the group boundary is reached. This methodology is a cornerstone of efficient SAS programming for summary data preparation.

Important Considerations and Best Practices

While the implicit **FIRST.** and **LAST.** variables are powerful tools for group processing, their correct usage depends on strict adherence to certain programming best practices. Ignoring these considerations can lead to inaccurate results or significant performance bottlenecks, particularly

with large datasets.

The Sorting Imperative: The single most crucial rule is that PROC SORT must always precede the DATA step containing the BY statement. If the records belonging to a group are not physically grouped together, the **FIRST.** and **LAST.** flags will operate based on spurious, non-contiguous groupings, yielding invalid results.
Multi-Variable Grouping: When defining groups using multiple variables in the BY statement (e.g., `BY Country Region;`), SAS creates a unique pair of **FIRST.** and **LAST.** variables for each BY variable. For instance, `FIRST.Country` indicates the start of a new country, while `FIRST.Region` indicates the start of a new region *within* the current country. Understanding this hierarchical dependency is crucial for complex grouping logic.
Performance Considerations: While vital for grouped processing, the PROC SORT step can be resource-intensive when dealing with massive datasets. Always verify if your data is already sorted by the required variable before running an unnecessary sort step to optimize performance.

Conclusion

The **FIRST.** and **LAST.** variables are fundamental components of advanced SAS programming, offering unparalleled control over data processing boundaries. By correctly integrating these implicit variables with the BY statement and ensuring your data is appropriately sorted, you unlock the ability to perform complex, iterative tasks--such as generating summaries, calculating running totals, or filtering unique records--with remarkable efficiency and clarity.

Mastering the application of **FIRST.** and **LAST.** is essential for moving beyond basic SAS data management. Incorporate these techniques into your workflow to produce robust, readable, and highly optimized code for all your grouped data analysis needs.

Additional Resources for SAS Programming

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