How to Use FIRST. and LAST. Variables in SAS

Authored by

stats writer

November 29, 2025

The FIRST. and LAST. variables are among the most powerful automatic variables available in the SAS programming environment. These variables are indispensable tools for data analysts who need to perform conditional processing based on group boundaries, calculate statistics, or restructure complex datasets.

Whenever a BY group is processed within a SAS DATA step, these two variables are automatically generated and assigned binary values (0 or 1). The presence of the `BY` statement signals to the SAS compiler that observations should be processed sequentially based on the values of the grouping variable. This mechanism allows for sophisticated control over iteration and data modification, paving the way for advanced data summarization and subsetting techniques.

Effectively using FIRST. and LAST. variables fundamentally depends on proper data preparation, specifically sorting the data by the grouping variable specified in the `BY` statement. If the input data is not sorted correctly, the results derived from these variables will be inaccurate, potentially leading to incorrect calculations or erroneous subsets. Throughout this guide, we will explore the precise definition of these variables and demonstrate practical applications using SAS code examples, showcasing how to identify group boundaries and efficiently select specific observations.

The **FIRST.** and **LAST.** automatic variables provide a straightforward method in SAS to flag the start and end of a defined group within a dataset. This flag is crucial for performing actions only once per group, such as initializing a counter or writing a summary record.

Understanding the Mechanics of FIRST. and LAST.

These variables are logical flags created dynamically during the execution of a SAS DATA step that includes a `BY` statement. They operate as temporary variables, meaning they are not added to the resulting output dataset unless explicitly specified. Their values are derived based on changes in the value of the corresponding `BY` variable from one observation to the next.

Here is a detailed breakdown of how each function assigns its value during the iterative processing of the DATA step:

FIRST.variable_name: This variable is automatically set to a value of **1** (True) only for the absolute **first** observation corresponding to a new value of the `BY` variable. For all subsequent observations within that same BY group, its value reverts to **0** (False). This flag is typically used to perform initialization tasks or select the first record of a group.
LAST.variable_name: Conversely, this variable is automatically set to a value of **1** (True) only for the absolute **last** observation corresponding to a specific value of the `BY` variable. Like its counterpart, it is set to **0** (False) for all prior observations within that group. This flag is ideal for

calculating final group summaries or selecting the last record.

It is important to remember that for a group consisting of only a single observation, both **FIRST.variable_name** and **LAST.variable_name** will simultaneously be assigned a value of 1. This is because the single observation is both the first and the last record of that unique group. Leveraging this characteristic is key to accurate data processing when dealing with grouped data in SAS.

Prerequisite: Sorting Data for Group Processing

The most critical requirement for utilizing FIRST. and LAST. variables correctly is ensuring that the input dataset is properly sorted by the variable referenced in the `BY` statement. If the data is not sorted, SAS will still process the `BY` groups, but these groups will be defined by sequential changes in the variable's value, which may not align with the logical groups intended by the user.

The standard procedure involves using the PROC SORT procedure prior to the DATA step where the `BY` statement is employed. Sorting ensures that all observations belonging to a specific group (e.g., all records for 'Mavs') are processed consecutively, allowing the automatic variables to accurately identify the start and end boundaries. This is non-negotiable for reliable group-based processing.

Failure to sort the data will not typically result in an error message from SAS, but it will lead to logical errors in the output. For instance, if records for 'Team A' appear, then records for 'Team B', and then more records for 'Team A', SAS will treat the second instance of 'Team A' records as a brand new BY group, setting `FIRST.team=1` again, thereby undermining the integrity of the group analysis.

Setting Up the Demonstration Dataset

To illustrate the functionality of these variables, we will use a hypothetical dataset named `my_data`, which contains statistics for several sports teams. This dataset includes a categorical variable (`team`) and two quantitative variables (`points` and `rebounds`). Notice that the data is initially entered in a structured format, but for group processing, the `team` variable will serve as our grouping key.

The following code creates the raw dataset and uses PROC PRINT to display its initial structure:

```
/*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

This resulting output shows the unsorted, raw input data. Our goal is to analyze this data based on the unique values of the **team** variable, identifying the first and last record for each team.

Example 1: Using FIRST. to Flag Group Starts

The **FIRST.** variable is incredibly useful when you need to execute a command or initialize a calculation only once for the starting record of every group. Common applications include setting counters to zero, initializing summary variables, or simply flagging the first entry to remove duplicates later. In this initial example, we will explicitly create a new variable, `first_team`, which will capture the binary value of `FIRST.team` for every observation.

First, we must sort the `my_data` dataset by the `team` variable. Then, in the subsequent DATA step, we use the `BY team;` statement, which automatically activates the creation of `FIRST.team`. By setting `first_team = first.team;`, we instruct SAS to store the value of the automatic variable into a permanent variable in the output dataset, `first_team`. This allows us to visually confirm exactly where each new group begins.

```
/*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_flagged;
set my_data;
by team;
first_team=first.team;
run;

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

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

Observe the resulting output table. The newly created **first_team** column successfully assigns the first observation for each team (Mavs, Rockets, Spurs) a value of **1**. All other records within those respective groups are assigned a value of **0**. This clearly delineates the beginning of each BY group.

Subsetting Data Using FIRST.

Beyond merely flagging records, the logical value of **FIRST.variable** can be used directly in an `IF` statement to subset the dataset, selecting only the first record of each group. This technique is extremely common when eliminating duplicate records or creating a distinct list of group identifiers. When `if first.team;` is used, SAS implicitly checks if `first.team = 1`, thereby retaining only the records that mark the start of a group.

The following streamlined code snippet demonstrates how to create a new dataset that exclusively contains the first observation for every unique team entry:

```
/*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;
set my_data;
```

```
by team;
if first.team;
run;

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

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

As clearly shown in the resulting table, the dataset `first_team` now only contains one record per team, effectively extracting the first instance of each unique team based on the criteria established by the **FIRST.** variable. This is a highly efficient method for deduplication or creating master reference lists.

Example 2: Using LAST. to Flag Group Ends

The **LAST.** variable functions identically to **FIRST.**, but it focuses on identifying the conclusion of a BY group. This is typically used when calculating final group statistics, such as a sum or average, which should only be output once the entire group has been processed. For example, if you were accumulating points across a team, you would only want to write the final total to the output dataset when `LAST.team` equals 1.

In this example, we once again ensure the data is sorted by team. We then create a new variable, `last_team`, which explicitly records the value of the automatic variable `LAST.team`. This allows us to observe which record marks the end boundary of each group.

```
/*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_flagged;
set my_data;
by team;
```

```
last_team=last.team;
run;

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

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

Upon reviewing the output, it is clear that the **last_team** column assigns a value of **1** only to the final observation associated with each team--the fourth observation for Mavs, the fourth for Rockets, and the fourth for Spurs. All preceding records within those groups are marked with **0**. This confirms the variable's function in correctly identifying the trailing boundary of the group.

Subsetting Data Using LAST.

Just as with **FIRST.**, the **LAST.** variable can be used directly in a conditional statement to select only the concluding observation of each group. This can be particularly useful if you only need the cumulative information or the final status of a sequence of records. By using `if last.team;`, we instruct SAS to process and retain only those records where the group defined by `team` ends.

The code below performs the necessary sorting using PROC SORT and then executes a DATA step to create a new dataset containing only the final observation for each team:

```
/*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;
set my_data;
by team;
if last.team;
run;

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

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

The resulting dataset confirms that only the final observation for each team has been retained. This powerful subsetting capability allows analysts to easily isolate endpoints, such as the highest recorded score, the latest date, or the final status value, provided the data is sorted by the relevant grouping and time variables.

Combining FIRST. and LAST. for Advanced Logic

The real power of these automatic variables emerges when they are combined in conditional logic. For instance, you can identify groups that consist of only a single observation by checking if `FIRST.variable` AND `LAST.variable` are both equal to 1. Conversely, you can exclude single-record groups by using `IF NOT (FIRST.variable AND LAST.variable);`.

This dual approach is essential for tasks like calculating the range of a variable within a group (using the first and last observation), or creating a summary table where certain actions (like printing group totals) are executed only upon reaching the final record of the group. Mastering the application of FIRST. and LAST. variables is foundational for efficient data manipulation in SAS programming.

Further SAS Data Step Applications

The principles demonstrated here using simple subsetting can be extended to complex analytical tasks. For example, using the combination of `BY group` processing and the automatic variables is the standard method for calculating running totals, cumulative sums, and inter-group comparisons within the SAS DATA step, offering flexibility often unmatched by standard PROC steps.

For those looking to deepen their expertise, exploring how to implement cumulative calculations using a `RETAIN` statement in conjunction with **FIRST.** variables is the logical next step. This allows a variable's value to persist across iterations of the DATA step, resetting only when a new group begins, thereby facilitating cumulative aggregation.

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