

The Power of the BY Statement

Paul Choate, California Department of Developmental Services, Sacramento, CA
Toby Dunn, Idea Integration, San Antonio, TX

Introduction

When used to its fullest potential, the BY statement can save time and effort, and add clarity and simplicity to SAS programs. It can make complex coding problems simple or accomplish in one pass what would otherwise require multiple passes of the data. Unfortunately the full power of the BY statement is often left unleashed, and instead is relegated to simple mundane everyday work. This paper will take you through the basics of the BY statement: the data step loop control statements SET, UPDATE, and MERGE as well as look at what the BY statement can do in procedures (PROCS). Along the way we will touch on sorting and how proper data structure will help you unleash the power that is the BY statement to gain overall efficiency in your code. The intended audience is beginner and intermediate level SAS programmers.

The BY statement is used by SAS to instruct the DATA step or PROCs to process SAS records in groups, rather than singly. It can be used whenever SAS data is ordered, or can be accessed in order through an index. In the DATA step this allows data to be summarized or reorganized according to a group structure. In PROC steps it allows SAS to process and present data in groups.

BY Statement Syntax and Usage

The basic syntax of the BY statement is the same throughout SAS, with the exception that the GROUPFORMAT option is only available in the DATA step.

```
BY <DESCENDING> var1 <...<DESCENDING> varn> <NOTSORTED> <GROUPFORMAT>;
```

In its simplest form the BY statement is followed by the list of variables on which the data is sorted. Data may be previously sorted, or sorted during processing with PROC SORT or PROC SQL with a GROUP BY statement. Once sorted, SAS data steps and procedures can take advantage of the ordered structure through use of the BY statement. For example the following data could be accessed with:

```
BY Sex Age Name;
```

NAME	SEX	AGE	HEIGHT	WEIGHT
Alice	F	13	56.5	84
Barbara	F	13	65.3	98
Carol	F	14	62.8	102.5
Judy	F	14	64.3	90
Jeffrey	M	13	62.5	84
Alfred	M	14	69	112.5
Ronald	M	15	67	133
Philip	M	16	72	150

Sort order is platform dependent based on the internal ordering of the platform character set, called the collating sequence. For example, most personal computers use the ASCII character set which is ordered 1, 2, 3, ... A, B, C, ... a, b, c ... IBM mainframes use the EBCDIC character set with order a, b, c, ... A, B, C, ... 1, 2, 3, ... Character code sets have 256 characters based on permutations of eight binary bit values, forming what is called a byte or a character. The SAS RANK function returns the position of a character in the collating sequence. Sorts are performed from the leftmost character to the rightmost character, as conventional in dictionaries or indexes. Collating sequences include all the printable characters, as well as non-printable characters such as tabs and line breaks. This paper refers to sorts on an ASCII platform, but the concepts are applicable on all supported platforms.

Sorts need not be in the order of the collating sequence, they may be also ordered as descending, in unsorted groups, or in unsorted groups based on a SAS format. If the ages were ordered in descending order the corresponding BY statement would be:

```
BY Sex DESCENDING Age Name;
```

NAME	SEX	AGE	HEIGHT	WEIGHT
Janet	F	15	62.5	112.5
Carol	F	14	62.8	102.5
Alice	F	13	56.5	84

Barbara	F	13	65.3	98
Philip	M	16	72	150
Alfred	M	14	69	112.5
Henry	M	14	63.5	102.5

If data is grouped on key variables, but not sorted, then NOTSORTED can be used. That is to say that the data need only be grouped and the groupings need not be in any kind of sorted order:

```
BY Age NOTSORTED;
```

Name	Age	Height	Weight
Carol	14	62.8	102.5
Judy	14	64.3	90
Janet	15	62.5	112.5
Ronald	15	67	133
Mary	15	66.5	112
Alice	13	56.5	84
Jeffrey	13	62.5	84

If a format can be used and the result from applying that format to the data will result in the data being grouped the GroupFormat option may be used. :

```
PROC FORMAT;
  VALUE $Initials 'A'-'B'='A'
                  'B'-'C'='B'
                  'C'-'D'='C'
                  ...
```

```
run;
```

```
DATA Class;
  SET Class;
  FORMAT Name $Initials.;
  BY Name GROUPFORMAT;
RUN;
```

NAME	AGE	HEIGHT	WEIGHT
Alice	13	56.5	84
Alfred	14	69	112.5
Judy	14	64.3	90
Janet	15	62.5	112.5
Jeffrey	13	62.5	84
William	15	66.5	112

DATA STRUCTURE

By now you have probably notice we have been talking an awful lot about processing data as groups. Reducing problems to groups of observations in reality just makes solving many problems simpler by allowing us to concentrate more on the business rules. The most important thing that will affect the By Statements and thus your ability to identify and use group processing is the structure of your data. This is often an over looked part of a SAS programmers education and you wont find a SAS class that deals with just data structures but knowing what a good data structure is and what is not will serve you well, and help you write better code beyond just By Group Processing.

In order to really understand what a good data structure is we need to first delve into what a dataset really is and its parts. To do this we need to look no further than data base land as these people have already done all the hard work for us. A dataset can be thought of as a collection of facts that represents the some part of the real world and the data are collected for some purpose. It may help to think a dataset as a mini world unto itself for now. Each piece of information can be thought of as a fact about this mini world. Now since we will want to do something with this data we will need to insure that it is organized in some fashion preferable in the most efficient fashion, thus we need to impose some rules that the data must follow to ensure it is organized correctly.

In a dataset variables can be thought of as attributes which describe something about the mini world we will call what they are describing an observation. Within this mini world we can break the variables into two classes: variable(s) that when their values are combined uniquely identify one observation or event we will call these Primary Key variable(s) and those who values when combined do not uniquely identify a observation these are called Non-Prime Keys. This means that all the variables are related to each other in some fashion and is usually shown as functional dependencies. Obviously having multiple observations with the same set of Primary Key values would in efficient because we would have serious redundancy in our mini world which causes many problems when say you are trying to identify which observation to update. So as a rule no two rows will have the same set of unique values for all variables.

VEHICLETYPE	MODEL	BRAND	YEAR	COLOR
Truck	1500	Chevy	2002	Blue
Truck	1500	Chevy	2002	Blue

Should be represented as

VEHICLETYPE	MODEL	BRAND	YEAR	COLOR
Truck	1500	Chevy	2002	Blue

Furthermore since each variable is describing a fact it is inefficient to have more than one variable trying to describe the same fact. In other words one should not have information in the variable names which can and should be facts themselves.

VEHICLETYPE	MODEL	BRAND	YEAR	RED	BLUE	GOLD
Truck	1500	Chevy	2002	1	2	3

This should be:

VEHICLETYPE	MODEL	BRAND	YEAR	COLOR	NUMSOLD
Truck	1500	Chevy	2002	Red	1
Truck	1500	Chevy	2002	Blue	2
Truck	1500	Chevy	2002	Gold	3

Lastly we should have a dataset where each variable has atomic values. This means that each value stores only one fact about that observation. Otherwise the data structure is said to have a relation within a relation. Having variables with atomic values ensures optimal efficiency when one needs only one of the values from this variable.

ID	NAME	ADDRESS	CLASSES
12	Jones, Tom	1501 WestOver JonesTown, 78901	{Science, Math, English}

Should be:

ID	FIRSTNAME	LASTNAME	ADDRESS	CITY	ZIPCODE	CLASS
12	Tom	Jones	1501 WestOver	JonesTown	78901	Science
12	Tom	Jones	1501 WestOver	JonesTown	78901	Math
12	Tom	Jones	1501 WestOver	JonesTown	78901	English

This example and the previous example correct method shows a dataset that is said to be in 1st Normal form with redundancies. While there are many different Normal Forms that the data can have almost all of the time 1st normal form will work. We will leave it to you the reader to investigate Normal Forms further on your own. This Normal Form of data creates the optimal data structure that allows for By Group Processing.

BY STATEMENT IN THE DATA STEP

The BY statement controls the operation of a SET, MERGE, MODIFY, or UPDATE statement in the DATA step, and provides two temporary variables for each BY variable: *FIRST.variable* and *LAST.variable*. The values are set to 1 when the first or last values in a group are present and 0 otherwise.

The *FIRST.variable* and *LAST.variable* values indicate whether an observation is:

- the first one in a BY group
- the last one in a BY group
- neither the first nor the last one in a BY group

- both first and last, as is the case when there is only one observation in a BY group.

```
SET class;
BY Sex Age;
```

SEX	AGE	FIRST.SEX	LAST.SEX	FIRST.AGE	LAST.AGE
F	13	1	0	1	0
F	13	0	0	0	1
F	14	0	0	1	0
F	14	0	0	0	1
F	15	0	0	1	0
F	15	0	1	0	1
M	13	1	0	1	1
M	14	0	0	1	0
M	14	0	0	0	1
M	15	0	0	1	0
M	15	0	0	0	1
M	16	0	1	1	1

The temporary FIRST.variable and LAST.variable are used to identify the breaks between groups during DATA step processing. The ordering of the variables is hierarchical, in this case Age is sorted within Sex. This is useful for many things, such as unduplicating the data or counting members.

This example shows how data can be unduplicated by Sex and by Age within Sex:

```
DATA Undup_Sex(KEEP=Sex) Undup_Age_Within_Sex;
  SET class;
  BY Sex Age;

  IF FIRST.Sex THEN OUTPUT Undup_Sex;
  IF FIRST.Age THEN OUTPUT Undup_Age_Within_Sex;
RUN;
```

```
Undup_Sex:  SEX
           F
           M
```

```
Undup_Age_Within_Sex:  SEX    AGE
                      F      13
                      F      14
                      F      15
                      M      13
                      M      14
                      M      15
                      M      16
```

The following example shows how data can be counted by Sex and by Age within Sex:

```
DATA Sex_Count(drop=Age Count_Age_Within_Sex)
  Age_Within_Sex_Count(drop=Count_Sex);
  SET class;
  BY Sex Age;

  IF FIRST.Sex THEN Count_Sex=0;
  IF FIRST.Age THEN Count_Age_Within_Sex=0;

  Count_Sex+1;
  Count_Age_Within_Sex+1;

  IF LAST.Sex THEN OUTPUT Sex_Count;
  IF LAST.Age THEN OUTPUT Age_Within_Sex_Count;
```

```
RUN;
```

```
Sex_Count: SEX      Count_Sex
           F         6
           M         6
```

```
Age_Within_Sex_Count:  SEX  AGE  Count_Age_Within_Sex
                       F    13      2
                       F    14      2
                       F    15      2
                       M    13      1
                       M    14      2
                       M    15      2
                       M    16      1
```

Variables on which data are sorted or indexed are called key variables. Using the power of the BY statement in the data step, key variables are used to process data in groups, including merging and interleaving datasets with the same keys.

Keys can be either unique or non-unique. For unique keys all FIRST.variable and LAST.variable temporary variables will have the value 1. In the result Age_Within_Sex_Count of the last example Sex is non-unique while Age is unique:

SEX	AGE	FIRST.SEX	LAST.SEX	FIRST.AGE	LAST.AGE
F	13	1	0	1	1
F	14	0	0	1	1
F	15	0	1	1	1
M	13	1	0	1	1
M	14	0	0	1	1
M	15	0	0	1	1
M	16	0	1	1	1

COMBINING DATASETS

So far we have considered the effects of the BY statement while reading a single dataset. In the data step, the BY statement is often used for combining data, either by interleaving, match-merging, updating, or modifying. Modifying a dataset is an advanced form of updating and is beyond the scope of this paper.

INTERLEAVING DATASETS

When a BY statement is used with a SET statement that specifies two or more datasets, the data step reads the two files simultaneously, alternating sequentially between the files based on the lowest level groupings of the BY variables. This has the advantage of maintaining the sort order of the data as it is processed.

For example, suppose there are two datasets, one for males and one for females, and both are sorted on Age. They can be interleaved into a single dataset sorted on Age and Gender. Since the values of Sex are unique within each dataset, the variable may be included on the BY statement even though it is not explicitly sorted.

```
PROC SORT DATA=Sashelp.Class OUT=Boys;
  BY Age;
  WHERE Sex='M';
RUN;

PROC SORT DATA=Sashelp.Class OUT=Girls;
  BY Age;
  WHERE Sex='F';
RUN;

DATA Class;
  SET Boys Girls;
  BY Age Sex;
RUN;
```

Now that we have seen a basic interleaving two data sets together let us now see if we cant use this basic technique to do some cool data processing. Consider the case where you need the sum of a variable by some group attached to the non aggregated data set. This will always require atleast two passes of the data, so our goal should not be how to limit the number of passes of the data to one but rather how we can reduce the amount and complexity of the code that does this. A few years back Howard Shreier wrote a paper entitled "Interleaving a Dataset with Itself: How and Why?". This paper showed that one can take this interleaving a data set and produce just what we are looking for, summed values attached to the base data set.

```
DATA Have;
INFILE Cards;
INPUT X Y $ @@;
CARDS;
1 A 2 A 3 A 4 A 5 B 6 B 7 B 8 B
;
RUN ;
```

```
Data Need ;
Set Have ( In = FirstPass )
      Have ( In = SecondPass ) ;
By X ;

If FirstPass Then Do ;
  If First.Y Then SumY = 0 ;
  SumY + Y ;
End ;
Else If SecondPass Then Output ;

Run ;
```

This Technique works because SAs will read all the data for a by group from the first instance of the data set Have (we labeled this FirstPass). It will then process this data summing Y. Then when the same by group of data is read again for the SecondPass we explicitly tell SAS to output the observation. Since the SumY + Y implicitly retains the value of SumY and we only reset this variable to 0 at the beginning of a By Group on the FirstPass of the data the value summed value will be in the PDV when we output the data to the Data Set Need on the SecondPass. What Howard had manage to do was to use a very basic technique I conjunction with the By statement in a very unique way there by produce code that is not only efficicnt but very readable and flexible.

MATCH-MERGING DATASETS

When a BY statement is used with a MERGE statement that specifies two or more datasets, the data step reads the two files simultaneously, merging observations from the two datasets based on the lowest level groupings of the BY variables. When merging two datasets usually one or both of the datasets should be unique at the lowest level of the sort groupings. In this case the unique observation is merged with each matching observation in the non-unique dataset. When merging more than two datasets, usually only one set at most should be non-unique on the BY variables. The unique observations are then duplicated across the non-unique merged observations.

The following Stock and Price datasets are sorted and match-merged on variable Item. The variable Item is unique in the Price dataset and is duplicated with Price across matching observations in Stock.

TITLE	ITEM
Castaways and Cutouts	CD
Picaresque	CD

Billy Liar	EP
The Tain	Single

ITEM	PRICE
CD	\$16.99
EP	\$10.99
Single	\$6.99

```
DATA Inventory;
  MERGE Stock Prices;
  BY Item;
RUN;
```

TITLE	ITEM	PRICE
Castaways and Cutouts	CD	\$16.99
Picaresque	CD	\$16.99
Billy Liar	EP	\$10.99
The Tain	Single	\$6.99

UPDATING DATASETS

The UPDATE statement is similar to the MERGE statement but only uses two datasets, a *master* dataset and a *transaction* dataset. UPDATE functions similarly to MERGE, but is used when missing transaction values are not wanted to overwrite existing values in the master dataset. In the UPDATE statement the master dataset is specified first and the transaction dataset second, and it is followed by a BY statement with the sorted or grouped variables on which to update. As with MERGE, the data step reads the two datasets simultaneously, updating observations from the master datasets with observations from the transaction dataset based on the lowest level groupings of the BY variables. When a transaction variable has a missing value, by default UPDATE does not overwrite the value in the master dataset, whereas the MERGE statement does.

The following master Inventory and transaction Price datasets are sorted and updated on variable Item. By default missing Price values do not overwrite Price values in matching Items of Inventory.

TITLE	ITEM	PRICE
Castaways and Cutouts	CD	\$16.99
Picaresque	CD	\$16.99
Billy Liar	EP	\$10.99
The Tain	Single	\$6.99

ITEM	PRICE
CD	\$17.99
EP	.
Single	\$8.99

```
DATA Inventory;
  UPDATE Inventory Prices;
  BY Item;
RUN;
```

TITLE	ITEM	PRICE
Castaways and Cutouts	CD	\$17.99
Picaresque	CD	\$17.99
Billy Liar	EP	\$10.99
The Tain	Single	\$8.99

Following is an example of the UPDATE statement being used to “flatten” a dataset that contains different data values for a key spread across several observations. The goal is to combine non-missing values into one record per key value. The master dataset structure is read, but OBS=0 stops the datastep from reading any data from it. The dataset is then updated as a transaction dataset, flattening multiple observations for one key into one observation.

```
DATA Scores;
  INPUT ID Sat_Math Sat_Verbal Act_Comp;
```

```

CARDS;
8188 560 . .
8188 . 540 .
8188 . . 12
8189 660 . .
8189 . 740 13
8189 . . .
;
RUN;

DATA Scores;
  UPDATE Scores(OBS=0) Scores;
  BY ID;
RUN;

ID    SAT_MATH  SAT_VERBAL  ACT_COMP
8188   560      540      12
8189   660      740      13

```

DO-LOOP OF WHITLOCK (DoW)

One of the data step techniques that usually rely heavily upon by group processing is the Do-Loop of Whitlock or more commonly called the DoW. Some time back Ian Whitlock answered a question on the SAS listserv, while the actual post is of little consequence to us what he did is. Ian wrapped the set statement inside of a Do Until loop. He used a By Statement to control this loop. Paul Dorfman later recognized the power that this construct brought to the programming tool box and started slowly started expounding upon this constructs use in many solutions to coding problems.

There are many slightly different forms of the DoW but the general form of the DoW is:

```

DATA ...;
  <Stuff done before break-event>;
  DO <Index Specs> UNTIL (Break-Event);
    SET A;
  <Stuff done for each record>;
  END;
  <Stuff done after break-event...>;
RUN;

```

While this set up may seem pretty tame it has great power when combined with the data step and the natural functionality of the data step. In a normal data step one observation is read in, something is done to that observations, and it is then written out. When we use By group processing in this context we tell SAS to specify groups of values for us and then we the programmer use the FIRST. and LAST. automatic variables to do certain things at certain times.

An example of this is when we want to sum a variables values and output one observation per by group.

```

DATA Have;
  INFILE Cards;
  INPUT X Y $ @@;
CARDS;
1 A 2 A 3 A 4 A 5 B 6 B 7 B 8 B
;
RUN;

DATA Example1 (DROP=X);
  SET Have;
  BY Y;
  SumX+X;

IF Last.Y THEN Do;
  OUTPUT;
  SumX=0;

```

```
END;
RUN;
```

This while this construct works there is nothing natural about it. In other words it relies on the programmer explicitly denoting when things happen before, during and after some break event, to do this FIRST. and LAST. BY variables are used with explicit output statements and resetting of certain variables to zero. The DoW on the other hand works with the natural functionality of the Data Step by isolating what happens between two consecutive break events, in the code above it the break events are By Groups, from what is to happen before and after the isolated code. While the break event usually is By Groups it does not have to be, it could be anything that triggers the Do-Loop to stop.

The following DoW example does the same thing as the above Data Step Example.

```
DATA Need;
DO UNTIL(Last.Y);
  SET Have;
  BY Y;
  SumX = SUM(OF SumX, X, 0 );
END;
RUN;
```

In the above example since there is no code before the Do-Loop the set statement inside of the Do-Loop starts reading observations into the Data Step and the values of X will summed until the last observation in the By Group is reached. When this happens the observation will be output to the dataset Need since there is no other SAS code to execute and no explicit output statement. Then control returns to the top of the data step and the do loop starts SAS reading observation from the point at which it last stopped. The X values are then summed for the following By Groups values. When the end of the by group is reached the Do-Loop stops and the observation is output to the dataset need. This process is repeated over and over again until there are no more observations in the input dataset.

Since there is no explicit output statement the implied output statement is used. Also there is no need to set the variable SumX back to zero for each by group. The reason for this lies in how the data step works, in the normal flow of a data step variable created by assignment statements get reset to missing with each iteration of the data step. In order to create a variable that accumulates from one observation to the next the variable has to have its value retained either explicitly or implicitly. With the DoW control never gets returned to the top of the data step until after all the observations of a by group have been read and the cumulative sum is already finished and outputted to the output dataset. Thus when implicit loop starts over again the value of SumX gets reset but it gets reset correctly when we need it to between by groups. This alleviates the programmer from having to remember to reset the cumulating variable to zero between by groups.

As a natural extension of the DoW one can use multiple DoW loops inside of a data step. Let's say one wanted to have the aggregated value for a variable attached to the base data.

```
DATA Need;
DO UNTIL(Last.Y);
  SET Have;
  BY Y;
  SumX = SUM(OF SumX, X, 0);
END;

DO UNTIL(Last.Y);
  SET Have;
  BY Y;
  OUTPUT;
END;
RUN;
```

Here we read the data twice the first time to get the value of X for each by group and a second time to get the observations outputted with SumX attached. All values for variables that are originally in the input dataset get over written by the second DoW's values. Again since the control of the data step hasn't been return to the top of the Step the values of SumX are not reset yet and since the second DoW doesn't do anything to the value of this variable it gets attached to each record that is outputted by the second DoW.

Proc SQL

We saw earlier that some procedures will accept only By Statements and others will accept both by and class statements. Proc SQL is no different however it use a Group By statement which can be thought of as a By statement. This Group By statement affects any and all summary statistic functions regardless of whether that summary function is in the select clause or having clause. This gives rise to some interesting looking code that may look strange but is very efficient overall.

Lets look at our example where we wanted to have the aggregated value for a variable attached to the base data.

```
DATA Have;
INFILE Cards;
INPUT X Y $ @@;
CARDS;
1 A 2 A 3 A 4 A 5 B 6 B 7 B 8 B
;
RUN ;
```

To achieve our goal the code is as follows:

```
Proc SQL ;
Create Table Need As
Select *, Sum( X ) As SumX
  From Have
   Group By Y ;
Quit ;
```

Now If we wanted to slightly modify this and only allow data that has a group sum of 15 or better but not have the summed value as a variable on the output data set:

```
Proc SQL ;
Create Table Need As
Select *
  From Have
   Group By Y
   Having Sum( X ) > 15 ;
Quit ;
```

Even though the summary function is in the Having clause and not written out to the final data set, SQL will use the Group By variables to calculate the statistics to be used.

CONCLUSION

The BY statement can be creatively in many ways to control the data step. It has been shown how to use the BY statement to manipulate and summarize data within groups, to merge and interleave data on key variables, and many other tasks. While the By Statement in and of itself does not provide much power to ones program, when used in conjunction with the data step techniques, procedures, and a proper data structure it can be very powerful in reducing the complexity and run time of your programs. While we have shown many advanced uses of the By statement remember they are all based off of the fundamental concept of using the By statement.

REFERENCES

SAS 9.1.3 XP Platform
SAS Institute Inc., Cary, NC

SAS OnlineDoc 9.1.3 for the Web

SAS Institute Inc., Cary, NC

"Retaining, Lagging, Leading and Interleaving Data" By Chang Chung and Toby Dunn
PharmaSug 2005

RECOMMENDED READING

SAS-L@LISTSERV.UGA.EDU
<http://listserv.uga.edu/archives/sas-l.html>

Documentation for SAS Products and Solutions
<http://support.sas.com/documentation/onlinedoc/index.html>



CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

Paul Choate, Senior Programmer Analyst
California Department of Developmental Services
1600 9th Street, Sacramento, CA 95818
Work Phone: (916) 654-2160
E-mail: pchoate@dds.ca.gov

Toby Dunn, Computer Guru
Stat Works Inc.
San Antonio, TX
Work Phone : (512) 786-4982
E-mail: tobydunn@hotmail.com

Join the SAS-L! @ listserv.uga.edu or Google Groups

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.