

# One Datum to Remember: RETAIN Statement

Jeanina Worden, Chunmei Yang

Pacific Data Designs. Inc, CA

## ABSTRACT

As PROC TRANSPOSE and PROC SQL are becoming more and more popular, are you sometimes wondering if the long list of variables transposed by PROC TRANSPOSE can be shrunk, or are you confused by ID variables, BY variables along with many of the to-be-transposed variables? Or do you often need to calculate summary information of variables but are stuck with the calculation of running totals or comparison of values across observations using PROC SQL? Don't forget one of the most useful DATA step tools - RETAIN statement. It can be used to carry forward values across observations, create constant subject level variables, count number of events per subject, transpose data from tall and thin to short and fat, re-order variables in analysis databases, and do other data manipulation across observations. A few common examples of the use of the RETAIN statement on clinical trials data will be illustrated in this paper including the use of the RETAIN statement in the LOCF method (last observation carried forward).

## INTRODUCTION

Since PROC TRANSPOSE and PROC SQL were introduced, they have been widely used to complete tasks involving transposition and variable assignment. While these are valuable tools, it seems that the RETAIN statement has been forgotten. In comparison, once you understand how RETAIN works, you will find it can be a very useful and efficient programming tool. This paper will cover some common SAS programming tasks such as carrying over variable values from previous observation to the next, identifying constant subject level values, creating a counter (e.g. number of visits per patient), accumulating variables across observations (e.g. calculating total number of tablets each patient took), transposing data from multiple records to single record per subject, and re-ordering variables in the data set.

## HOW DOES RETAIN STATEMENT WORK?

DATA step processing has two phases: compilation and execution. During compilation, SAS checks code for syntax errors, establishes an area of memory called the program data vector (PDV) and assigns required attributes to variables. Then during the execution phase, by default SAS sets any variables that are not read from a SAS data set to missing when it starts a new iteration of the DATA step. If a RETAIN statement is used, SAS will not assign all the variables in the retain list to missing in PDV. Instead the values from previous iteration will be kept to next iteration. The values will only be changed if INPUT or assignment statements are encountered.

RETAIN statement has the following form:

```
RETAIN variable-list(s) <initial-value(s) >;
```

e.g. RETAIN x;

```
RETAIN x 0;
```

```
RETAIN x y z;
```

```
RETAIN x y z 0;
```

```
RETAIN x 1 y 2 z 3;
```

In the above example, x, y and z are variable names and 0-3 are initial values.

Without the RETAIN statement, SAS assigns all the variables to missing in the program data vector before it reads a new data row as shown below.

SAS code -reading data without RETAIN statement:

```

data tests;
  input ptid $ visit;
  datalines;
A    1
A    2
A    3
;
run;

```

The data reading process will be as following:

Table 1 Reading data without RETAIN statement

	PTID	VISIT
1 <sup>st</sup> row data – before reading	.	.
1 <sup>st</sup> row data – after reading	A	1
2 <sup>nd</sup> row data – before reading	.	.
2 <sup>nd</sup> row data – after reading	A	2
3 <sup>rd</sup> row data – before reading	.	.
3 <sup>rd</sup> row data – after reading	A	3

If the RETAIN statement is added to the above DATA step, the values for both PTID and VISIT at the beginning of each iteration, which are highlighted, are not assigned missing values any more. Values from the previous iteration have been retained as shown in Table 2 (highlighted).

SAS code -reading data with RETAIN statement:

```

data tests;
  retain ptid visit;
  input ptid $ visit;
  datalines;
A    1
A    2
A    3
;
run;

```

Table 2 Reading data with RETAIN statement

	PTID	VISIT
1 <sup>st</sup> row data - before reading	.	.
1 <sup>st</sup> row data - after reading	A	1
2 <sup>nd</sup> row data - before reading	A	1
2 <sup>nd</sup> row data - after reading	A	2
3 <sup>rd</sup> row data - before reading	A	2
3 <sup>rd</sup> row data - after reading	A	3

In Table 2, initial values before reading for both PTID and VISIT were assigned as missing values, which are the same as Table 1. But other values before reading in Table 2 were retained from previous iteration. This feature can be used to copy values over to the next observation and further data manipulation based on the retained values can be performed. The detailed use of RETAIN statement for different programming requirements is illustrated in the next section.

## COMMON USE OF RETAIN STATEMENT

### 1. CARRY THE VALUES FROM PREVIOUS ITERATION OVER TO NEXT ITERATION

LOCF (Last Observation Carried Forward) method has been widely used to take into account missing data in data analysis. The value from the last non-missing observation is carried forward to replace the missing value in the next observation. For example, the following data set includes some test results for two patients. PTID is the unique identifier of each patient. VISIT identifies the visit number and VISDATE is the visit date. RESULT is the test result of a certain clinical test. DOSE is the dose of the trial medication each patient takes. Suppose there should be 5 visits for each patient. Some patients miss some visits and other patients may have all the visits in the database but the test results are missing. LOCF method is used to fill the missing test results.

Example 1:

#### SAS code to create the data set TESTS

```
data tests;
  input ptid $ visit visdate result dose;
  datalines;
A   1   39114   50   100
A   2   39142   40   40
A   3   39173   .   30
A   4   39203   30   15
A   5   39234   30   15
B   1   38991   67   120
B   2   39022   63   60
B   5   39052   .   45
;
run;
```

A template data set is created first for all the possible visits of all the patients.

```
proc sort data=tests out=allpts(keep=ptid) nodupkey; by ptid; run;

data template;
  set allpts;
  by ptid;

  do visit=1 to 5;
    output;
  end;

run;
```

Then the template is merged with TESTS to ensure that there is one record for each of the 5 visits for each patient. To fill the missing values, RETAIN statement can be used in the same DATA step as in the following code.

```
proc sort data=tests; by ptid visit; run;
proc sort data=template; by ptid visit; run;

data locf;
  merge tests template;
  by ptid visit;

  retain newresult;
  if first.ptid then newresult=.;
  if result ^=. then newresult=result;
run;

proc print data=locf; title 'Use of RETAIN statement in LOCF'; run;
```

## SAS output of PROC PRINT

Use of RETAIN statement in LOCF					
ptid	visit	visdate	result	dose	newresult
A	1	39114	50	100	50
A	2	39142	40	40	40
A	3	39173		30	40
A	4	39203	30	15	30
A	5	39234	30	15	30
B	1	38991	67	120	67
B	2	39022	63	60	63
B	3				63
B	4				63
B	5	39052		45	63

The new variable NEWRESULT is the variable with missing values filled by the LOCF method. RETAIN statement specifies that the value of NEWRESULT is retained from one observation to the next observation unless an INPUT or assignment statement is encountered. For each patient, NEWRESULT is assigned a missing value to the first visit to prevent the last test result from being copied from one patient to the next. Then NEWRESULT equals the original test result if the original result is not missing. Otherwise if the original test result is missing, NEWRESULT keeps its value from the previous visit. As it can be seen, the test result for patient A Visit 3 is filled by the result of patient A visit 2, and for patient B, the test results of visit 3 and 4 are filled by the result of visit 2. The data set needs to be sorted first in order to use FIRST. PTID.

Note that variables that are read with a SET, MERGE, MODIFY or UPDATE statement are automatically retained. In the above example, it will not make difference if PTID and VISIT are added to RETAIN statement.

## 2. IDENTIFY CONSTANT SUBJECT LEVEL VARIABLES

Frequently constant subject level variables such as demographic and baseline character variables need to be identified and assigned to all the observations of each patient for data analysis. For instance if we are interested in the changes of the test results from follow up visits to screening visit, first the test result from screening visit needs to be assigned to all the observations.

Example 2:

SAS code:

```
proc sort data=tests; by ptid ; run;
data subj_var (keep=ptid visit result screen change);
  set tests;
  by ptid;

  retain screen;
  if first.ptid then screen=result;
  change=result-screen;

run;

proc print data=subj_var noobs; title 'Test Results Changes From Screening';
run;
```

## SAS output of PROC PRINT

Test Results Changes From Screening				
ptid	visit	result	screen	change
A	1	50	50	0
A	2	40	50	-10
A	3	.	50	.
A	4	30	50	-20

A	5	30	50	-20
B	1	67	67	0
B	2	63	67	-4
B	5	.	67	.

In this example, the variable SCREEN identifies the test result of a patient at screening visit. As can be seen from the above example, once the values of a variable are retained, computation or comparison to other variables across observations becomes very easy and straightforward. Another example would be to calculate the total doses each patient takes as discussed in Example 3.

### 3. ACCUMULATE VALUES ACROSS OBSERVATIONS

The following code creates a running total of doses (TOTDOSE) for each patient by accumulating the doses from each visit. If the last observation for each patient is output to a data set, we will get the total doses for each patient.

Example 3:

SAS code

```
proc sort data=tests; by ptid visit; run;

data sumup (keep=ptid visit dose totdose);
  set tests;
  by ptid visit;

  retain totdose;
  if first.ptid then totdose=0;
  totdose=totdose+dose;

  /*if last.ptid then output; */ *Adding this statement will output a data
  set with total doses for each patient;
run;

proc print data=sumup noobs; title 'Running Total Doses for Each Patient';
run;
```

### SAS output of PROC PRINT

Running Total Doses for Each Patient			
ptid	visit	dose	totdose
A	1	100	100
A	2	40	140
A	3	30	170
A	4	15	185
A	5	15	200
B	1	120	120
B	2	60	180
B	5	45	225

### 4. CREATE A COUNTER

Similarly, a counter variable can be created to identify the sequential number of the visits. Suppose that in TESTS data set, there is an unscheduled visit, which is coded as 99. This visit is not always the last visit of a patient. A sequential visit number (VISITSEQ) needs be created to order all the visits including the unscheduled visit based on the visit date.

Example 4:

SAS code:

```
data tests2;
  input ptid $ visit visdate result dose;
```

```

datalines;
A   1   39114   50   100
A   2   39142   40   40
A   3   39173   .   30
A   4   39203   30   15
A   5   39234   30   15
B   1   38991   67   120
B   2   39022   63   60
B  99   39035   50   50
B   5   39052   .   45
;
run;

proc sort data=tests2; by ptid visdate; run;

data counts ;
  set tests2;
  by ptid visdate;

  retain visitseq;

  if first.ptid then visitseq=0;
  visitseq=visitseq+1;

  /*if last.ptid then output; */ *Adding this statement will output a data
  set with total number of visits for each patient;

run;

proc print data=counts noobs; title 'Sequential Number of Visits for Each
Patient';
run;

```

#### SAS output of PROC PRINT

Sequential Number of Visits for Each Patient					
ptid	visit	visdate	result	dose	visitseq
A	1	39114	50	100	1
A	2	39142	40	40	2
A	3	39173	.	30	3
A	4	39203	30	15	4
A	5	39234	30	15	5
B	1	38991	67	120	1
B	2	39022	63	60	2
B	99	39035	50	50	3
B	5	39052	.	45	4

#### 5. TRANSPOSE DATA FROM MULTIPLE OBSERVATIONS PER PATIENT TO ONE OBSERVATION PER PATIENT

Sometimes a short and fat data set with all the information for each patient in one observation is preferred for data analysis. The following SAS code transposes data set TESTS to a short one with one observation per patient.

Example 5:

SAS code:

```

proc sort data=tests; by ptid visit; run;

data tests_tran (keep=ptid result1-result5 dose1-dose5 ) ;
  set tests ;
  by ptid visit;

```

```

array resulta{*} result1-result5 ;
array dosea{*} dose1-dose5;

retain resulta dosea ;
if first.ptid then do;

do i=1 to dim(resulta);
  resulta{i} =.;
  dosea{i} =.;
end;
end;

do j=1 to dim(resulta);
  if visit=j then do;
    resulta{j}=result;
    dosea{j}= dose;
  end;
end;

if last.ptid then output;
run;

proc print data=tests_tran noobs; title 'Transposed Data'; run;

```

#### SAS Output of PROC PRINT

Transposed Data										
ptid	result1	result2	result3	result4	result5	dose1	dose2	dose3	dose4	dose5
A	50	40	.	30	30	100	40	30	15	15
B	67	63	.	.	.	120	60	.	.	45

RESULT1- RESULT5 and DOSE1-DOSE5 are the new transposed variables corresponding to each visit. Array statements are used to represent these variables in the following assignment statement. For the first visit of each patient, these variables are assigned to missing. RESULT1 equals RESULT when VISIT=1. And RESULT2 equals RESULT when VISIT=2 and so on. Once these five variables are assigned non-missing values, the values remain across observations till the last record for this patient. If a patient misses some test results, missing values are assigned. Finally using LAST.PTID the transposed information for each patient is output to the data set TESTS\_TRAN.

#### 6. ASSIGN THE ORDER OF THE COLUMNS

Often a certain order of the variables in the data set is required, such as for the convenience of viewing the data using PROC PRINT or SAS System Viewer, and clinical data submission to clients and FDA. RETAIN statement is one of the most efficient ways to re-order variables. As the example shows below, variables in the data set COUNTS are re-ordered to PTID VISITSEQ VISDATE VISIT DOSE RESULT. Note that the RETAIN statement has to be placed before the SET statement so that RETAIN statement is read first and the variables are compiled in PDV in the same order as listed in the RETAIN statement. Otherwise if the SET statement is read first, the variables will be compiled in the same order as in the COUNTS data set.

Example 6:

#### SAS code

```

data counts2;
  retain PTID VISITSEQ VISDATE VISIT DOSE RESULT;
  set counts;
run;

Proc print data=counts2 noobs; title 'Re-ordered Columns'; run;

```

### SAS output of PROC PRINT

Re-ordered Columns					
PTID	VISITSEQ	VISDATE	VISIT	DOSE	RESULT
A	1	39114	1	100	50
A	2	39142	2	40	40
A	3	39173	3	30	.
A	4	39203	4	15	30
A	5	39234	5	15	30
B	1	38991	1	120	67
B	2	39022	2	60	63
B	3	39035	99	50	50
B	4	39052	5	45	.

### COMPARISON TO PROC TRANSPOSE AND PROC SQL

In SAS there are always multiple ways to program a specific requirement. Another convenient way to transpose data is to use PROC TRANSPOSE. If transposing only one variable, PROC TRANSPOSE is efficient. If there are many variables to be transposed, RETAIN statement is preferable, as you need to use one PROC TRANSPOSE to transpose each variable and then sort all the transposed data and merge them together again. To transpose the same data as in Example 5, PROC TRANSPOSE needs to be used twice and then the transposed data sets need to be merged together.

#### SAS Code

```
proc sort data=tests; by ptid visit; run;

proc transpose data=tests out=result_tran(drop=_name_) prefix=result;
  by ptid;
  var result;
  id visit;
run;

proc transpose data=tests out=dose_tran(drop=_name_) prefix=dose;
  by ptid;
  var dose;
  id visit;
run;

proc sort data=result_tran; by ptid; run;
proc sort data=dose_tran; by ptid; run;

data both_tran;
  merge result_tran dose_tran;
  by ptid;
run;

proc print data=both_tran noobs; title 'Transposed Data'; run;
```

If there is a long list of variables to be transposed, using RETAIN statement in a DATA step can be much more efficient. In addition, other programming tasks such as creating new variables based on the transposed variables can be combined in the same DATA step if required. In fact the above Example 2 – 5 can be combined within one DATA step. This is left to the readers for practice.

PROC SQL is another way to do some of the tasks in the above examples. To calculate the total doses and total visits for each patient, PROC SQL can be used to accomplish this task very efficiently using the following code.

#### SAS code

```
proc sql;
  create table sqldata as
  select ptid, visit, sum(dose) as totdose, count(visit) as totvisit
  from tests
  group by ptid;
quit;
```

Compared to DATA step, PROC SQL does not require the data to be sorted first. Also it calculates the summary variables TOTDOSE and TOTVISIT, and assigns them to all the observations of the patient using one SELECT statement. It is another efficient way to re-order the variables in a data set as well. However DATA step using RETAIN statement is more flexible and more efficient when manipulating the data observation by observation, such as carrying forward values across observations and creating running totals. When joining multiple data sets, especially when the data sets have non-matching observations and inner join and outer join need to be used, programming using PROC SQL can be lengthy and confusing. Using the DATA step is much more straightforward. In addition multiple data sets can be created from one DATA step while each PROC SQL can only create one table with one CREATE TABLE statement.

#### CONCLUSION

Though the examples in this paper can be accomplished with other tools such as PROC TRANSPOSE and PROC SQL, the use of RETAIN can be a more efficient way of completing these tasks. Using a single RETAIN statement takes advantage of the PDV to initialize a new variable prior to being in data, with or without a value, and can “hold” the value from a previous record until a change occurs. So when you need to carry forward values across observations, create a counter, make an accumulated variable, or re-order the variables in a data set, remember RETAIN statement!

#### REFERENCES

SAS Institute, Inc., 2004, SAS Online Doc, Version 9.1. Cary, NC; SAS Institute, Inc.

Worden, Jeanina (2007), “Skinny to Fat: In search of the “Ideal” dataset”, Proceedings of the Thirty-Second Annual SAS Users Group International Conference.

Delwiche, Lora D. and Susan J. Slaughter. 2003. The little SAS Book: A Primer, Third Edition, Cary, NC: SAS Institute Inc.

#### ACKNOWLEDGMENTS

We would like to thank Frank Han and Ryan Kimes for their review and comments on a draft of this paper.

#### CONTACT INFORMATION

We welcome your comments and questions. You can contact the authors at:

Jeanina Worden, Chunmei Yang  
Pacific Data Designs, Inc.  
353 Sacramento Street, Suite 800  
San Francisco, CA 94111  
(415) 766-0660  
[jworden@pdd.net](mailto:jworden@pdd.net) / [cyang@pdd.net](mailto:cyang@pdd.net)

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.  
Other brand and product names are trademarks of their respective companies.