PharmaSUG2011 – Paper CC15

QUICK – READY SET RETAIN, AND MAYBE RESET!

Lisa Fine, United Biosource Corporation, Ann Arbor, MI

ABSTRACT

The RETAIN statement is one method that SAS® programmers commonly use for making comparisons across observations. One source of misunderstanding around the RETAIN statement centers around how long a value is retained and the ability or need to reset retained variables in many circumstances. This paper clarifies, through examples, how the RETAIN statement overrides the default behavior of a DATA step and maintains a variable's values until the value is reset. The scope of the paper is to demonstrate the use of the RETAIN statement in conjunction with assignment statements.

INTRODUCTION

The RETAIN statement "Causes a variable that is created by an INPUT or assignment statement to retain its value from one iteration of the DATA step to the next"[1] This is in contrast to the default DATA step behavior, which is, "Without a RETAIN statement, SAS automatically sets variables that are assigned values by an INPUT or assignment statement to missing before each iteration of the DATA step." [1]

What does that mean in practical terms? It means that the SAS programmer can now easily make comparisons and derivations across observations. For instance, if we can create a variable to retain a patient's baseline weight on every record, we can calculate the change from baseline, by subtracting baseline weight from current weight at any visit for which weight was measured. Another use of RETAIN might involve retaining the most recent visit's weight, so we can use this value in the case that weight was not measured for a particular visit.

One point of confusion has arisen with regard to how long a variable retains a value when the RETAIN statement is used. Jones and Whitlock[2] remind the user that RETAIN does not mean that a variable must retain its value indefinitely. Rather, it is a request to not automatically set the variable to missing at the top of each implied DATA step loop. The programmer can always execute commands that change the value of the RETAIN variable.

In its most basic form (RETAIN of a single variable) the syntax for the RETAIN statement is

```
RETAIN <variable name <initial_value;
```

Example 1: **`RETAIN x 0;`** *Assigns the variable x an initial value of 0*

Example 2: **`RETAIN x .;`** *Assigns the variable x an initial value of . (missing), x will be written to data set*

Example 3: **`RETAIN x;`** *Also Assigns the variable x an initial value of . (missing), x will not be written to data set unless an initial value is assigned elsewhere*

HOW LONG IS A 'RETAIN' VALUE RETAINED? A VALUE IS RETAINED UNTIL…VOILA, IT IS RESET

The following examples further clarify how to use the RETAIN statement, particularly with regard to the need to sometimes reset the RETAINed variables. This first example demonstrates how the values of a RETAIN variable change at various steps in a program.

EXAMPLE 1

Here is the research problem: *Patients are expected to summarize their symptoms in a daily diary. The programmer is tasked with identifying if a patient had missed days, i.e. days in which the patient did not make an entry.*

The data set used is a small data set with three variables. The data is already sorted by PATIENT (only 1001 is shown) and Diary Entry Date.

FIGURE 1 - DATA

REC	PATIENT	DIDATE
1	1001	01FEB2009
2	1001	02FEB2009
3	1001	04FEB2009
4	1001	06FEB2009

Key:
REC = *Record Number*
PATIENT = *Patient ID*
DIDATE = *Diary Entry Date*

Here is the logic behind the program to be developed.

PRE-PROGRAM LOGIC

Overall point of using RETAIN for this example: *By retaining previous date completed we can verify that the previous entry was one day ago, i.e. a missed day is indicated where lapse is >1.*
Main Steps:
• Create a RETAIN variable 'XPREVDT' that will hold the changing values of previous date.
• For the first occurrence of each PATIENT set the RETAIN variable to current DIDATE so the previous PATIENT's information does not carry forward.
• Determine days since previous diary entry (LAPSE).
• Reset XPREVDT to the current date to be used as previous date for the next observation.

Below is the simple program that will count the gap in days between diary entries. Two variables will be created in the program. XPREVDT, the RETAIN variable will hold the most recent diary date, and LAPSE will compute the difference between current and most recent diary entry.

FIGURE 2 - PROGRAM

```
DATA DIARY1;
   SET DIARY;
   BY PATIENT DIDATE;
   FORMAT XPREVDT DATE9.;

1  RETAIN XPREVDT;

2A IF FIRST.PATIENT THEN
      DO;
         XPREVDT=DIDATE;
      END;

**GET LAPSE;
ELSE
   DO;
2B    IF NMISS(DIDATE,XPREVDT)=0 THEN LAPSE = DIDATE - XPREVDT;
      ELSE LAPSE=.;
2C    XPREVDT=DIDATE;  **RESET TO CURRENT DATE;
   END;
RUN;
```

Log output created by PUT statements after each programming section (e.g., 1, 2A, 2B, 2C) provides a view of how records are impacted at each step. In other words, with the PUT statements you can see the interim values of the RETAIN variable XPREVDT. (In contrast the PROC PRINT would show only the final XPREVDT results, i.e. the Step 2A and 2C results).

WALKING THROUGH EXAMPLE 1:

FIGURE 3 LOG OUTPUT shows how records 1-4 in our data set process at each step of the above program. The processing starts with Record 1 (REC 1). At Step 1, XPREVDT for REC 1 is set to missing. At Step 2A, REC 1's XPREVDT value is set = to DIDATE. Note that REC 1 is patient 1001's first record so it follows through the 'IF FIRST.PATIENT' program logic.

FIGURE 3 – LOG OUTPUT – Patient 1001

PROGRAM STATEMENT AND DESCRIPTION			LOG OUTPUT			
1	`RETAIN XPREVDT;`	Initialize XPREVDT to missing	**REC=1**	didate=01FEB2009	xprevdt=.	lapse=.
2A	`IF FIRST.PATIENT THEN DO; XPREVDT=DIDATE; END;`	1st patient occurrence - Reset XPREVDT to DIDATE	**REC=1**	didate=01FEB2009	xprevdt=01FEB2009	lapse=.

Starting with patient 1001's REC=2, the records follow through the 'ELSE' (i.e. not FIRST.PATIENT) program logic. For example at step 1 in the program, XPREVDT for REC 2 is not reset to missing and therefore maintains the previous XPREVDT value of 01FEB2009 (until reset at 2C). After Step 2B, REC 2's LAPSE shows a value of 1, the difference between DIDATE and XPREVDT. After step 2C, XPREVDT is equal to DIDATE, just as the program requested.

1	`RETAIN XPREVDT;`	Do not reinitialize XPREVDT to missing	**REC=2**	didate=02FEB2009	xprevdt=01FEB2009	lapse=.
2B	`ELSE DO; IF NMISS…THEN LAPSE=DIDATE-XPREVDT; ELSE LAPSE=. END;`	Calculate lapse	**REC=2**	didate=02FEB2009	xprevdt=01FEB2009	lapse=1
2C	`XPREVDT=DIDATE;`	Reset XPREVDT to DIDATE	**REC=2**	didate=02FEB2009	xprevdt=02FEB2009	lapse=1

1	`RETAIN XPREVDT;`	Do not reinitialize XPREVDT to missing	**REC=3**	didate=04FEB2009	xprevdt=02FEB2009	lapse=.
2B	`ELSE DO; IF NMISS…THEN LAPSE=DIDATE-XPREVDT; ELSE LAPSE=. END;`	Calculate lapse	**REC=3**	didate=04FEB2009	xprevdt=02FEB2009	lapse=2
2C	`XPREVDT=DIDATE;`	Reset XPREVDT to DIDATE	**REC=3**	didate=04FEB2009	xprevdt=04FEB2009	lapse=2

1	`RETAIN XPREVDT;`	Do not reinitialize XPREVDT to missing	**REC=4**	didate=06FEB2009	xprevdt=04FEB2009	lapse=.
2B	`ELSE DO; IF NMISS…THEN LAPSE=DIDATE-XPREVDT; ELSE LAPSE=. END;`	Calculate lapse	**REC=4**	didate=06FEB2009	xprevdt=04FEB2009	lapse=2
2C	`XPREVDT=DIDATE;`	Reset XPREVDT to DIDATE	**REC=4**	didate=06FEB2009	xprevdt=06FEB2009	lapse=2

The above example shows how XPREVDT, the RETAIN variable changed as it was reset via assignment statements. XPREVDT served as a temporary storing variable so that day lapse between current and previous date could be calculated. For this example, any lapse >1 indicates a missed entry.

Following are key points associated with each section of the program (designated by the boxed numbers/letters).

KEY POINTS:

1 `RETAIN XPREVDT;`

(This statement could have been placed later in the program and had the same result)

The RETAIN initialization occurs only once, during compile time. This is in contrast to assignment statements that are potentially executed for each record. For example, see REC 1 versus RECs 2, 3 and 4. Only REC 1's XPREVDT was initialized to missing as specified in the RETAIN statement. At REC 2, for example, XPREVDT = 01FEB2009, the previous record's DIDATE. This was the purpose of using RETAIN for this example. We want XPREVDT to hold the previous record's value so we can calculate time lapse, rather than follow the default SAS behavior of resetting XPREVDT to missing at the beginning of the DATA step for each new record.

2A

```
IF FIRST.PATIENT THEN DO;
   XPREVDT=DIDATE;
END;
```

The conditional assignment statement based on whether it is the patient's first record resets the value of XPREVDT to DIDATE. **The purpose of this reset is to prevent one patient's value from being carried forward to the next patient**. Because we have put the RETAIN in place, we need to explicitly re-initialize XPREVDT every time a new patient is encountered.

2B

```
ELSE DO;
   IF NMISS(DIDATE,XPREVDT)=0 THEN LAPSE = DIDATE - XPREVDT;
   ELSE LAPSE=.;
END;
```

This is the conditional assignment statement for records that are not the first occurrence for a patient. **Now that we have previous entry date (XPREVDT) on the same record as current date (DIDATE) we can easily calculate the difference, LAPSE, between the two entry dates.**

2C

```
XPREVDT = DIDATE;
```

At step 2C we **reset XPREVDT to capture the current DIDATE, which will become the next record's 'previous' date**. It is okay to reset XPREVDT at this point because LAPSE between current and previous entry has already been calculated for observations where applicable (i.e. not FIRST.PATIENT).

EXAMPLE 2

WHAT HAPPENS WHEN RETAIN VARIABLES ARE NOT RESET?

While the answer seems obvious, forgetting to reset RETAINed variables is a common mistake when learning to use the RETAIN statement. FIGURE 4 contains the diary data used earlier, plus records for two additional patients.

FIGURE 4: DIARY DATA SET

REC	PATIENT	DIDATE
1	1001	01FEB2009
2	1001	02FEB2009
3	1001	04FEB2009
4	1001	06FEB2009
5	1002	17FEB2009
6	1002	18FEB2009
7	1002	19FEB2009
8	1003	12FEB2009
9	1003	16FEB2009
10	1003	19FEB2009

EXPLICIT RESETS

There are two explicit resets of the RETAIN variable XPREVDT in our program, 2A and 2C.

2A

Resets XPREVDT to DIDATE for each first occurrence of an patient in the data set

```
IF FIRST.PATIENT THEN
   DO;
      XPREVDT=DIDATE;
   END;
```

2C

Applies to non-first occurrences of PATIENT and resets XPREVDT to DIDATE, AFTER lapse has been calculated for the current observation.

```
XPREVDT = DIDATE;
```

It is useful to observe what happens when the resets are excluded. FIGURE 5 shows the final PROC PRINT output when both resets are correctly included. FIGURES 6 and 7 respectively, display the results when resets 2A, and 2C are excluded.

FIGURE 5 – CORRECT RESULTS WHEN BOTH RESETS (2A AND 2C) ARE IN PLACE

FINAL DATA - DIARY - ALL RESETS

REC	patient	didate	LAPSE
1	1001	01FEB2009	.
2	1001	02FEB2009	1
3	1001	04FEB2009	2
4	1001	06FEB2009	2
5	1002	17FEB2009	.
6	1002	18FEB2009	1
7	1002	19FEB2009	1
8	1003	12FEB2009	.
9	1003	16FEB2009	4
10	1003	19FEB2009	3

FIGURE 6 – INCORRECT RESULTS WHEN RESET 2A IS NOT IN PLACE

FINAL DATA - DIARY - WITHOUT FIRST RESET

REC	patient	didate	LAPSE
1	1001	01FEB2009	.
2	1001	02FEB2009	1
3	1001	04FEB2009	2
4	1001	06FEB2009	2
5	1002	17FEB2009	**11**
6	1002	18FEB2009	1
7	1002	19FEB2009	1
8	1003	12FEB2009	**-7**
9	1003	16FEB2009	4
10	1003	19FEB2009	3

LAPSE - Previous PATIENT'S values were carried forward. For example, PATIENT 1001's most recent entry 06FEB2009 was subtracted from PATIENT 1002's current date 17FEB2009 = 11 days.

FIGURE 7 – INCORRECT RESULTS WHEN RESET 2C IS NOT IN PLACE.

FINAL DATA - DIARY - WITHOUT SECOND RESET

1	1001	01FEB2009	.
2	1001	02FEB2009	1
3	1001	04FEB2009	**3**
4	1001	06FEB2009	**5**
5	1002	17FEB2009	.
6	1002	18FEB2009	1
7	1002	19FEB2009	**2**
8	1003	12FEB2009	.
9	1003	16FEB2009	4
10	1003	19FEB2009	**7**

LAPSE - XPREVDT is never reset after FIRST.PATIENT record. E.g., All of 1001's LAPSES are based on current didate minus 01FEB2009. All of 1002's LAPSES are based on current didate minus 17FEB2009. All of 1003's LAPSES are based on current didate minus 12FEB2009.

CASES IN WHICH A VARIABLE IS AUTOMATICALLY RETAINED

The RETAIN statement is used for newly created variables, that is for variables read from a raw data set or created newly by an assignment statement. The reason RETAIN is used with new variables is because these variables, by default are initialized to missing at the beginning of each data loop and RETAIN is meant to counteract this behavior. In contrast, there are several types of variables that are automatically retained. These include temporary array variables (variables from arrays named _TEMPORARY_), automatic variables (e.g., _N_, _ERROR_) and SAS data set variables (variables read from a SAS data set using SET, MERGE, UPDATE, and MODIFY). It would be redundant to use the RETAIN statement in these cases where variables are automatically retained or in other words are not initialized before each new observation is read. The following section discusses the initialization process for NEW, and SAS data set variables. The reader can refer to the cited references for more detail on the other types of automatically retained variables.

Here is a comparison of the Initialization process as it occurs for New Variables (WITHOUT a RETAIN statement) (8A) versus SAS data set variables (8B).

INITIALIZATION PROCESS FOR…

A. **Creating a NEW variable** - In this case the variable is initialized to missing before EACH RECORD is read (**FIGURE 8A**). NEW variables include variables input from RAW data sets and variables created in most assignment statements. (There are exceptions such as assignments with the use of RETAIN or SUM statements and DO loops.)

B. A SAS data set variable (i.e. called with SET, MERGE, UPDATE, MODIFY) - In this case the variable is initialized to missing before the FIRST RECORD ONLY (**FIGURE 8B**).

FIGURE 8A - Raw Data Set / New Variable

patient=	_N_=1
patient=1001	_N_=1
patient=	_N_=2
patient=1001	_N_=2
patient=	_N_=3
patient=1002	_N_=3
patient=	_N_=4
patient=1003	_N_=4

FIGURE 8B - SAS Data Set read with SET, MERGE, UPDATE, or MODIFY

patient=	_N_=1
patient=1001	_N_=1
patient=1001	_N_=2
patient=1002	_N_=3
patient=1003	_N_=4

WHY IS THE INITIALIZATION PROCESS RELEVANT WHEN NO RETAIN STATEMENT IS INVOLVED?

As mentioned earlier, SAS data set variables are automatically retained. Therefore, the need for resetting values may become pertinent when values are not automatically being reset (or re-initialized). The example below (9A) shows how unexpected results can occur when a user attempts to modify an existing SAS variable. Figures 9B and 9C show work-arounds for this situation if it is necessary to use/modify the existing variable.

Both data sets ONE and TWO, below and are sorted by DAY. FIGURES 9A-9C below show different coding attempts to accomplish the same result, which is to reassign FLAG a value of 'N' when TYPE = 2.

DATA ONE

DAY	FLAG
1	Y
2	Y
3	Y

DATA TWO

DAY	TYPE
1	3
1	4
2	2
2	4
3	1
3	2
3	1

Three programs (FIGUREs 9A, 9B, and 9C), and their corresponding PROC PRINT output are shown below. Note that two coding options (9B, 9C) produce the correct result. In contrast, 9A provides incorrect results.

WHICH ONE DOESN'T BELONG?

CONCATENATING DATA SETS

FIGURE 9A
No Reset

```
DATA THREE;
   SET ONE
       TWO;
   IF TYPE=2 THEN FLAG='N';
RUN;
```

Obs	DAY	TYPE	FLAG
1	1	.	Y
2	2	.	Y
3	3	.	Y
4	1	3	
5	1	4	
6	2	2	N
7	2	4	N
8	3	1	N
9	3	2	N
10	3	1	N

INCORRECT

FIGURE 9B
Rename Original FLAG so FLAG becomes a new variable

```
DATA THREE;
   SET ONE(RENAME=(FLAG=OLDFLAG))
       TWO;
   IF TYPE=2 THEN FLAG='N';
   ELSE FLAG=OLDFLAG;
RUN;
```

FIGURE 9C
Reinitialize variables with New DATA Step Values

```
DATA THREE;
   SET ONE
       TWO;
RUN;

DATA THREE;
   SET THREE;
   IF TYPE=2 THEN FLAG='N';
RUN ;
```

Obs	DAY	TYPE	FLAG
1	1	.	Y
2	2	.	Y
3	3	.	Y
4	1	3	
5	1	4	
6	2	2	N
7	2	4	
8	3	1	
9	3	2	N
10	3	1	

CORRECT

What went wrong with 9A? The problem in 9A occurs when we attempt an assignment statement with FLAG, which already exists in data set ONE. As mentioned above, other than for the first record, SAS data set variables are automatically retained (not reset to missing) across iterations. Since FLAG was first read from data set ONE, its value will be retained between iterations. When we set flag to 'N' when TYPE=2, because FLAG is a SAS variable the 'N' from the previous observation is retained; there are no FLAG values in TWO to overwrite it and it is never reset. Note that even though a new assignment is being made FLAG is not a 'new' variable and the new variable rules do not apply.

In contrast, 9B and 9C work because FLAG gets reset. In 9B, FLAG becomes a new variable because DATA ONE's FLAG gets renamed to OLDFLAG. The new variable rules now apply to FLAG, and FLAG gets reset to missing before each observation. In 9C, when data set THREE is SET, each record of the data set now has a FLAG value, some of which are missing but it is important to note that missing is a valid value. For example, when FLAG is set to 'N' in observation 6 it is overwritten by missing when observation 7 is read.

WHAT ABOUT WITH A BY STATEMENT?

The first occurrence of each BY value will be re-initialized but after the first BY value occurrence, SAS variables will be retained. Example 9A would be incorrect regardless, but had we interleaved data sets one and two (for example, sorted and SET them BY DAY) we would have gained one correct observation, because at the first DAY 3 (BY variable) observation, FLAG would get reinitialized to missing and would not retain the final DAY 2's 'N'**.** (This process does not show in PUT statements but it can be demonstrated in PROC PRINTS, such as FIGURE 10A). In FIGURE 10A, which shows an example of MERGING datasets, it can be seen that we gained one correct record over 9A. Observation 5 did not retain Observation 4's 'N' because at the first DAY 3 (BY variable) occurrence FLAG got reinitialized to missing. The rest of the FIGURE 10 results are the same story as for the FIGURE 9 series. FIGURE 10A leads to incorrect results and 10B and 10C provide correct results for the same reasons as in the FIGURE 9 series.

MERGING DATA SETS

FIGURE 10A
No Reset

```
DATA THREE;
   MERGE ONE
         TWO;
   BY DAY ;
   IF TYPE=2 THEN FLAG='N';
RUN;
```

Obs	DAY	TYPE	FLAG
1	1	3	Y
2	1	4	Y
3	2	2	N
4	2	4	N
5	3	1	Y
6	3	2	N
7	3	1	N

INCORRECT

FIGURE 10B
Rename Original FLAG so FLAG becomes a new variable

```
DATA THREE;
   MERGE ONE(RENAME=(FLAG=OLDFLAG))
         TWO;
   BY DAY ;
   IF TYPE=2 THEN FLAG='N';
   ELSE FLAG=OLDFLAG;
RUN;
```

FIGURE 10C
Reinitialize variables with New DATA Step Values

```
DATA THREE;
   MERGE ONE
         TWO;
   BY DAY ;
RUN;

DATA THREE;
   SET THREE;
   IF TYPE=2 THEN FLAG='N';
RUN ;
```

Obs	DAY	TYPE	FLAG
1	1	3	Y
2	1	4	Y
3	2	2	N
4	2	4	Y
5	3	1	Y
6	3	2	N
7	3	1	Y

CORRECT

CONCLUSION

The RETAIN statement overrides the default process and prevents the value of a variable from being reinitialized to missing at the beginning of the data loop. This allows for many additional capabilities, in particular manipulations across observations. However, because by using the RETAIN statement we have turned off the automatic reinitialization feature for the RETAINed variable it is then necessary to explicitly reset the variable at times. In the case of SAS variables read with SET, MERGE, UPDATE, AND MODIFY, variables are automatically retained. Attempting to modify an existing SAS variable often requires the user to create a new variable or modify the variable in a separate DATA step in order to avoid unwanted effects of the automatic retain.

REFERENCES

- [1]SAS Institute Inc. (2011) SAS® *9.2 Language Reference: Dictionary, Fourth Edition*, Cary, NC: SAS Institute, Available at http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/viewer.htm#a000214163.htm
- [2]Jones, Robin and Whitlock, Ian (2002), Missing Secrets, *Proceedings of the 2002 Southeast SAS Users Group Conference*, *PS05*.
- Gorrell, Paul (1999), The RETAIN Statement: One Window into the SAS® Data Step, *Proceedings of the 1999* Northeast SAS Users Group Conference, BT064.
- Henderson, Don and Rabb, Merry (1988), The SAS Supervisor, *Proceedings of the 1988 Northeast SAS Users Group Conference.*
- Lewis, Ginger and Whiteis, Grace (2010-2011), (SAS Technical Support) email correspondence
- Virgile, Bob (1999), How MERGE Really Works, *Proceedings of the 1999 Northeast SAS Users Group Conference*, AD155.
- Whitlock, Ian (1997), A SAS® Programmer's View of the SAS Supervisor, *Proceedings of the 1997 SAS Users Group International* 22 Conference, Cary, NC: SAS Institute Inc.

ACKNOWLEDGMENTS

The author would like to thank Carol Matthews, Paul Slagle, and Andrew Newcomer for their review of this paper and for their continual mentoring. Thanks very much to Venky Chakravarthy for his feedback on how to improve this paper. The author thanks the cited authors in the REFERENCES section for providing a good basis for understanding this topic.

CONTACT INFORMATION

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

Lisa Fine
United Biosource Corporation
2200 Commonwealth Blvd., Suite 100
Ann Arbor, MI 48105
Work Phone: (734) 994-8940 x1616
Fax: (734) 994-8927
E-mail: lisa.fine@unitedbiosource.com
Web: www.unitedbiosource.com