What Does RETAIN Really Do?
John R Whittington, Mediscience Services Ltd., Buckingham, UK

ABSTRACT

The RETAIN statement is commonly used to provide a link between successive iterations of the SAS® DATA step, or for its incidental effect of ordering / re-ordering variables. Its effects are simple, but often clouded by suggestions that it presents hidden hazards. The question of what the RETAIN statement does, may do, and does not do is reviewed, together with consideration of the importance of the positioning of a RETAIN statement within a DATA step. Alternative methods of carrying values of variables from one DATA step iteration to another, using macro variables, temporary arrays or the LAGn function are also considered briefly.

INTRODUCTION

Although the function of the RETAIN statement of the SAS® language is essentially very straightforward, it has been a source of a surprising number of questions and heated debates on the SAS-L internet discussion group. In particular, concerns are sometimes voiced that it has effects which simply do not exist, and there is also sometimes uncertainty about the importance of the position of a RETAIN statement within a DATA step.

The intrinsic, implicit, iterative behaviour of the SAS DATA step is fundamental to the SAS language; the DATA step processes only one observation at a time and then starts afresh with the next observation, the majority of variables being 'set to missing' at the start of each iteration. Whilst this approach facilitates the processing of data sets of unlimited size, one consequence is that the DATA step has no intrinsic memory of the value of variables in previous iterations of the DATA step.

The RETAIN function is one method which is commonly used to carry over values from one DATA step iteration to another. The functionality of this statement is discussed in detail, followed by a brief consideration of other methods available to transfer variable values between different iterations of a DATA step.

TRANSFERRING VALUES OF VARIABLES BETWEEN DATA STEP ITERATIONS

There are several ways in which the programmer may allow SAS to retain knowledge of variable values from previous iterations of a DATA step, which include:

- use of the RETAIN Statement
- use of macro variables
- use of temporary arrays
- use of LAGn functions

The RETAIN statement permits single values of variables to be carried forward from one iteration of the DATA step to the immediately following iteration, whilst the other approaches have the potential to allow values to be transferred from one iteration to any subsequent one.

WHAT RETAIN ALWAYS DOES

The one, and only, thing which the RETAIN statement always does is to prevent variables being 'set to missing' at the start of each iteration of a DATA step, thereby allowing SAS to 'remember' values of variables from one iteration of a DATA step to the next. A variable which is RETAINed (or implicitly RETAINed - see below) will keep the same value through any number of DATA step iterations, unless and until that value is explicitly changed (e.g. by assignment, or data read with INPUT, SET etc.). Hence, without a RETAIN statement, the code:

```
data one ;
  pre_set = new_var ;
  set test ;
  new_var = from_set * 10 ;
r
```

...will result in the following output, indicating that the variable new_var, created by assignment, is 'set to missing' at the start of each DATA step iteration:

```
PRE_SET  FROM_SET  NEW_VAR
10 100
20 200
30 300
40 400
50 500
```

Adding a RETAIN statement causes the value of new_var to be retained (not 'set to missing') until that value is changed by the next assignment:

```
data one ;
  retain new_var ;
  pre_set = new_var ;
  set test ;
  new_var = from_set * 10 ;
r
```

...results in output ....

```
PRE_SET  FROM_SET  NEW_VAR
10 100
20 200
30 300
40 400
50 500
```

WHAT RETAIN DOES NOT NEED TO DO

There are many situations in which a RETAIN statement is not needed, since the values of several categories of variables are automatically retained from one iteration of the DATA step to the next (i.e. they are not 'set to missing' at the start of each iteration), even if there is no RETAIN statement relating to them. Categories of such automatically retained variables include the following:

- variables read with SET, MERGE or UPDATE statements
  - even if those statements are never executed, e.g.:
    ```
    if 0 then set mydata ;
    ```
• variables whose values are assigned in a sum statement.
  For example, when accumulating the total value of a variable across observations, the code:
  
  ```
  x + a;
  is entirely equivalent to:
  retain x 0;
  x = sum(x, a);
  ```

• elements of a temporary array
• automatic variables etc. (e.g. _N_)

It is also sometimes forgotten that the ‘resetting to zero’ of variables (which RETAIN prevents) is something that happens at the start of each iteration of the DATA step - which is not necessarily the same as the start of an observation. In particular, if the programmer takes explicit control of the looping through observations, using a DO loop, then the entire data set may be processed within just a single iteration of the DATA step, as in the following code:

```
data one;
do i = 1 to 10;
  set test;
  [ other statements here ]
  output;
end;
stop;
run;
```

Since programme execution does not return to the start of the DATA step between observations, no ‘resetting to missing’ occurs, so that all variables within the DATA step are effectively RETAINed, even in the absence of a RETAIN statement. Indeed, if the programmer wishes to emulate the behaviour of the implicit DATA step loop (without RETAIN), it is necessary to explicitly code a ‘setting to missing’ operation for all variables at the start of the DO loop.

**WHAT RETAIN SOMETIMES DOES**

Depending upon the nature of the variables referenced, and the positioning of a RETAIN statement within a DATA step, there are a few other things (desirable and undesirable) that a RETAIN statement may do:

**Determine the initial value for a variable**

The syntax of the RETAIN statement allows it to specify an ‘initial value’ - i.e. its value at the start of the very first iteration of the DATA step. That value will be retained, through all iterations of the DATA step, unless and until the value is changed - e.g. by assignment, reading a new value with INPUT, SET etc. Hence, if the variable x does not exist in the data set being SET, then the code:

```
data one;
  retain x 1234;
  set mydata; /* does not contain variable x */
run;
```

... results in variable x having a value of 1234 for every observation in the data set. If that value is changed (by assignment or data which is read), then the new value will be retained through all subsequent iterations of the DATA step unless/until changed again. Hence the code:

```
data one;
  retain x 1234;
  set mydata;
  if _N_ ge 6 then x = 5678;
run;
```

... would result in the variable ‘x’ having the value of 1234 for the first 5 iterations (observations) and a value of 5678 for the sixth and all subsequent observations.

**Create and determine some attributes of a variable**

If a RETAIN statement assigns an initial value to a variable, and if that variable has not previously been defined (e.g. created by assignment or an INPUT statement, or read with a SET, MERGE or UPDATE statement), then the RETAIN statement results in creation of variable with a type (numeric or character) and length (if character) as indicated by the specified initial value. If the variable has already been defined prior to the RETAIN statement, then RETAIN cannot alter the type or length of the variable (see below).

In the absence of an initial value, RETAIN will not, in itself, result in creation of a variable, even if it has not previously been defined. Hence, the code:

```
data one;
  retain a 4 b this_is_21_chars_long c;
run;
```

... results in creation of variable ‘a’ as a numeric one of (default) length 8 and ‘b’ as a character one of length 21 (each with the specified initial values) - but does not result in variable ‘c’ being created (no initial value specified). If variable ‘b’ had been previously defined as numeric, then an error condition would arise; if it had previously been defined as character with a length less than 21, then the initial value given would be truncated to fit the already-defined length.

**Determine the ordering of variables**

If the first mention of a variable in a DATA step occurs in the ‘element list’ of a RETAIN statement, then the order of variables in that list will determine their order in the Program Data Vector (PDV) and hence in any output data set. This is true whether the variables are created within the DATA step (by assignment) or read using INPUT, SET, MERGE or UPDATE statements.

When used for this purpose, RETAIN should be placed before any variables are defined (hence usually the first statement in a DATA step) and generally should be used without any specification of initial values. In this way, the RETAIN statement does not itself establish the type or length of variables, thereby avoiding the risk of any unwanted errors or conflicts.

**Cause error conditions due to variable type conflicts**

If a RETAIN statement attempts to assign an initial character value to a variable already defined as numeric, or vice versa, then an error condition will arise (no implicit type conversion will take place) and the DATA step will terminate. Taking the example of variables ‘num’ and ‘char’ (numeric and character respectively) being read with SET, the following logs show what happens if RETAIN is used (unnecessarily, for a variable read with SET) with an incorrect type of initial value, before or after the statement (SET) which establishes the variable type:
It is therefore important to be sure that any 'initial values' specified in a RETAIN statement are consistent with the variable type as defined either before or after the RETAIN statement.

WHAT RETAIN NEVER DOES

The most common unfounded fear expressed during some discussions about the RETAIN statement is that it may result in missing data being overwritten by a value that has been carried forward from the previous iteration of the DATA step.

This is simply not true. All that RETAIN does is prevent the value of the variable being set to 'missing' at the start of the DATA step iteration. Immediately a new value for that variable is created during the current iteration (whether by assignment, or from an INPUT, SET, MERGE or UPDATE statement), the variable in question takes that new value, even if that new value is 'missing'.

If one wishes missing data to be replaced by a value carried forward from a previous observation, then this has to be explicitly coded - usually by RETAINing the value in a 'scratch' variable which is not in the data set being read, for example:

```sas
data one (drop = old_x);
  set mydata;
  if x = . then x = old_x;
  old_x = x;
run;
```

WHEN THE POSITION OF RETAIN MATTERS

Early in its description of RETAIN, the SAS documentation says that RETAIN can appear anywhere in a DATA step, because it is not an executable statement. However, as the documentation indicates subsequently, this may cease to be true in at least two situations:

When _ALL_, _CHARACTER_ or _NUMERIC_ are used as the 'element list'.

In this situation, only variables which have already been defined before the RETAIN statement are actually retained. Hence, with the code:

```sas
data one;
  set test; /* has only vars x, y, z */
  a = x;
  b = y;
  retain _ALL_;
  c = z;
run;
```

... variables 'a' and 'b' would be retained, along with (implicitly) all those variables read with the SET statement, but variable 'c' would not be retained.

When RETAIN is being used to order / re-order variables

When RETAIN is being used to establish or change the order of variables in the PDV (hence output data sets), the first mention of the variables concerned must be in the 'element list' of that RETAIN statement. That generally means that RETAIN is placed as the first statement in the DATA step.

RETAIN WITHOUT AN ‘ELEMENT’ LIST

If the RETAIN statement is used without an 'element list', this has the same effect as if RETAIN _ALL_ had been placed at the very end of the DATA step (i.e. all variables in the DATA step would be RETAINed), even if that RETAIN statement was placed before some of the variables were defined. The positioning of RETAIN statement which has no 'element list' is therefore always unimportant.

OTHER METHODS OF TRANSFERRING VALUES OF VARIABLES BETWEEN DATA STEP ITERATIONS

Some of the alternative approaches to transferring values of variables between DATA step iterations will now be considered briefly:

Transfer via a macro variable

The value of a variable may be stored in a macro variable using the CALL SYMPUT routine. Although references to the macro variable (e.g. &myvar) will not be resolved during execution of the DATA step, these values may be recovered at any point during execution of the DATA step (not necessarily in the same iteration) using the SYMGET function. Hence, the functionality of RETAIN can be achieved with code such as:

```sas
%let mvar = '.'; /* clear any previous value */
data one;
  pre_set = input( symget('mvar'), best12.);
set test;
call symput('mvar', from_set);
run;
```

.... which gives output:

```
PRE_SET FROM_SET
.
10  20
20  30
30  40
40  50
```

In contrast with the use of RETAIN, this approach is not restricted to transfer of values between consecutive iterations of the DATA step. The value can be stored during any iteration, and then recovered during any subsequent iteration, provided a new value has not been assigned to the macro variable in the meantime.
Transfer via a temporary array

Similarly, elements of a temporary array can be used in exactly the same fashion as a macro variable for temporary storage of a variable value across iterations of the DATA step. The following code uses a single array element to achieve the same as the macro variable in the previous example:

```sas
data one;
  array arr (1) _temporary_;  
  pre_set = arr(1);
  set test;
  arr(1) = from_set;
run;
```

Use of the LAGn function

The intrinsic LAGn function of the SAS language creates a ‘stack’ of ‘n’ storage locations, the values of which ‘survive’ iterations of the DATA step. Each time the LAGn function is called, the value at the top of the stack is returned, and all elements of the stack are moved up one position, the current value of the variable in question being added to the bottom of the stack.

Hence, if the LAGn function is called once during each iteration of the DATA step, it will each time return the value of the variable concerned during the nth previous iteration - i.e. LAG1(x) will return the value of (x) during the previous iteration, whist LAG2(x) will return the value which was held by x during the penultimate iteration, and so on, for any value of n. Hence, given an input data set which provides values for the variable x, the code:

```sas
data one;
  set test;
  x_lag1 = lag1 (x);
  x_lag2 = lag2 (x);
run;
```

will produce output:

<table>
<thead>
<tr>
<th>X</th>
<th>X_LAG1</th>
<th>X_LAG2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>.</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

Great care is needed in the use of the LAGn function because of the way it works, which is often not fully considered. In particular, to obtain the functionality that is ‘usually’ required, it is essential that the LAGn function is called once, and only once, during each and every iteration of the DATA step (observation); use of LAGn in conditional statements (which may not be executed for every iteration) is a common cause of unwanted effects.

CONCLUSION

The functionality of the RETAIN function has been reviewed and other methods of transferring variable values between DATA step iterations have also been considered.