PharmaSUG 2012 - Paper TA05

Using SAS Colon Effectively

Sudhir Singh, EMD Serono, Rockland, MA

ABSTRACT

SAS® colon is not clearly documented or indexed in the SAS® manual. Although the colon is one of the most powerful useful symbol operators available to the programmer, some of its features are rarely found in typical SAS® code. In this paper I will explain how colon can be used as a operator modifier, label indicator, a format modifier, a keyword component, a variable name wildcard, an array bound delimiter or a special log indicator. Mastering these usages can give your code needed functionality and/or efficiency lift.

INTRODUCTION

The **colon** (**:**) is a punctuation mark consisting of two equally sized dots centered on the same vertical line. In computing, the colon character is represented by ASCII code 58, (HTML &#58;). The colon is quite often used as a special control character in URLs, programming languages, in the path representation of several file systems (such as HFS), and in many operating system commands. Several programming languages use the colon for various purposes. Though colon has many different uses in SAS® language it is not well documented. Colon is not properly indexed in SAS® onlinedoc, system help and printed SAS® manual. From the scattered documentations and publications this paper collected seven types of colon usage.

1. Operator modifier
2. Variable name wildcard
3. Keyword component
4. Array bound delimiter
5. Special log indicator
6. Label indicator
7. Format modifier

Some of the usage can improve coding efficiencies while other provides unique and necessary capacities for various circumstances. All the examples in this paper are tested in SAS version 9.1.3 and 9.2

1. OPERATOR MODIFIER

In SAS, the colon (:) can be used in conjunction with all of the comparison operators (=, >, <, >=, <=, ne, in, gt, lt) to compare prefix's. The colon transforms the meaning of comparison operator to '**begin with**'.

Consider the following examples, given the following surnames:

- Bevan
- Bosley
- Bowen
- Burden
- Bush

Consider the following data step statements using the colon modifier:

```
if surname =: 'B' then do;
```

In this case the colon transform the meaning of '=' to 'begin with'. This will find all 5 surnames.

In SAS, character strings must be adjusted to the same length before they can be compared. When SAS compares character strings without the colon modifier, it pads the shorter string with blanks to the length of the longer string before making the comparison. When SAS compares character string with the colon modifier after the operator, it truncates the longer string to the length of the shorter string. This feature of the colon modifier makes the comparison of a character string's prefix po**s**sible.

```
if surname =: 'Bo' then do;
```
It will find Bosley and Bowen only.

```
if surname >=: 'Bo' then do;
```
It will find Bosley, Bowen, Burden and Bush only.

```
if surname <: 'Bo' then do;
```
It will find Bevan only.

```
if surname ne: 'Be' then do;
```
It will find Bosley, Bowen, Burden and Bush only.

```
if surname in: ('Be','Bu') then do;
```
It will find Bevan, Burden and Bush only.

So, given the list of surnames – Tom, Tomlinson, Tomson , the following conditional statement will return the results Tom and Tomson

```
if surname =: 'Tomson' then do;
```

The surprising result here is that surname='Tom' actually meets this condition. In performing this comparison SAS has determined that the value of 'surname' is shorter than the right hand side of the equation, and so the right hand side get's truncated to the same length.

2. VARIABLE NAME WILDCARD

A colon following a variable name prefix selects any variable whose name starts with that prefix. This ability of the colon along with some simple naming standards enables the programmer to manage temporary variables better, format many variables quicker, determine unknown number of variables, clean up macro generated datasets, and shorten the code for variety of PROCS. For example -

```
data ADLB;
set  lb:;
```

This DATA step reads all the data sets in the WORK library that begin with LB. Also, when the programmer coded this step, he/she did not need to know how many dataset are read, only that he/she wants to read all of the dataset with a particular prefix. Both colon and dash lists also work with the MERGE statement.

When you process data within a DATA step, you often need to perform a variety of calculations. In this process, you may create variables that you do not wish to save in the output data set. The task of keeping track of them and DROP them from the new dataset becomes much more difficult. However, if we start all of our temporary variable names with the same distinct prefix such as TEMP, then it becomes very simple to get rid of these variables in the output data set we can easily write the following code.

```
Drop  TEMP: ;
```

This will drop any variable whose name start with 'TEMP'

```
Total = sum (of aval:);
```

This function will sum all the variables with the prefix 'aval' .

This same strategy could also be used in the reverse way. You could prefix all the variables that you intended to KEEP and use the colon suffix to KEEP them in the output data set. The other advantage with this method is that you can go back and edit your data step, adding and removing variables, without having to change the DROP or KEEP statements.

```
array  col  {*}   col:;
```

The above code creates an array for all ‘col’ variables. This could be very helpful when we have to process unknown number of variables. For example, if we have to process multiple number of observation per subject to one observation per subject. We can transpose the data but the numbers of parameters are different for each subject.

We can know easily use the do loop along with the above array statement to process such data.

```
Do i=1 to dim(col):

   ….;

End;
```

So the colon has enabled us to perform the necessary computations without ever knowing the number of observations per subject.

3. KEYWORD COMPONENT

A macro variable can be created from a column value in PROC SQL by specifying the INTO clause. A colon (:) is used in conjunction with the macro variable name being defined.

```
PROC SQL NOPRINT;

       SELCT PRODNAME, PRODCOST INTO : PRODNAME,   :PRODCOST
FROM PRODUCTS; QUIT;

%PUT &PRODNAME  &PRODCAST;
```

This structure turns the values of a selected field into a string of a macro variable for later use. A macro variable created via SELECT INTO: is an efficient way. For example it can be used to count number of observation-

```
proc sql noprint;
  select count(*) into :nobs from test;
quit;

%PUT &nobs;
```

4. ARRAY BOUND DELIMITER

We can now use a method called array processing to perform the same tasks for a series of related variables within a dataset. The variables involved in the processing are related by arrays. An array is a temporary grouping of SAS variables that are arranged in a particular order and identified by an array-name; its definition is avalid in the same data step.

By default in SAS, the subscript in each dimension of an array ranges from 1 to n, where n is the number of elements in that dimension. In an array statement, the lower bound can be omitted if its value is 1. An array with 2 dimension ranging from 1 to 3 and 1 to 6, respectively can be declared as:

```
Array test {3,6} test1- test8;
```

Using a formal bounded Array Declaration, we can declare this same array as

```
Array test {1:3,1:6}  test1-test18;
```

In a bounded array declaration, the colon is used to separate the lower and upper bounds of an array dimension. The bounded array declaration is useful in defining array dimension which are not based on 1. In the following case

Defining the array bounds according to year numbers makes the code more readable and reusable.

```
Array sales {1980:2000) sales 1980-sales2000;
```

5. SPECIAL LOG INDICATOR

A colon can be combined with keyword ERROR, NOTE or WARNING in a %PUT statement to generate customized error, note or warning text in SAS log. These particular user-defined log strings have the same color as the system generated error, note and warning messages. In the %PUT statement, the keyword must be the first word after %PUT, must be in upper case, and must be followed immediately by a colon or hyphen.

The following statements

```
%put ERROR: The variable TRT is not in the dataset;
%put NOTE:  The dataset ADSL contain 300 observations;
%put WARNING:  The dataset ADSL contain duplicates;
```

Generate these colored strings in the log:

```
ERROR: The variable TRT is not in the dataset;
NOTE:  The dataset ADSL contain 300 observations;
WARNING:  The dataset ADSL contain duplicates;
```

This feature can make customized error, warning or note text more readable in SAS log.

6. LABEL INDICATOR

A colon after a string signals the string as a label and the statement(s) after the colon as the labeled statement(s). Any statement(s) within the data step can be labeled, although no two labels in a data step can have the same name. The statement label identifies the destination of either a GO TO statement, a LINK statement, the HEADER= option in a FILE statement, or the EOF= option in an INFILE statement.

In this example, if Stock=0, the GO TO statement causes SAS to jump to the statement that is labeled reorder. When Stock is not 0, execution continues to the RETURN statement and then returns to the beginning of the DATA step for the next observation.

```
data Inventory Order;
  input Item $ Stock @;
    /* go to label reorder: */
  if Stock=0 then go to reorder;
  output Inventory;
  return;
    /* destination of GO TO statement */
  reorder: input Supplier $;
  put 'ORDER ITEM ' Item 'FROM ' Supplier;
  output Order;
  datalines;
milk  0 A
bread 3 B
;
```

7. FORMAT MODIFIER

SAS colon as an input/output modifier is documented in SAS manuals. The INPUT statement reads raw data from instream data lines or external files into a SAS data set. You can use list input, column input, and formatted input and named input styles, depending on the layout of data values in the records. SAS colon is used in list input.

LIST INPUT

List input uses a scanning method for locating data values. Data values are not required to be aligned in columns but must be separated by at least one blank (or other defined delimiter). List input requires only that you specify the variable names and a dollar sign ($), if defining a character variable. You do not have to specify the location of the data fields.

An example of list input follows:

```
data scores;
   length name $ 12;
   input name $ score1 score2;
   datalines;
Riley 1132 1187
Henderson 1015 1102
;
```

The **:** (colon) format modifier enables you to use list input but also to specify an informant after a variable name, whether character or numeric. SAS reads until it encounters a blank column, the defined length of the variable (character only), or the end of the data line, whichever comes first. Though the data step continues reading until it reaches the next blank column, it truncates the value of a character variable if the field is longer than its formatted length.

The following is an example of the : and ~ format modifiers. You must use the DSD option in the INFILE statement. Otherwise, the INPUT statement ignores the ~ format modifier.

```
data scores;
   infile datalines dsd;
   input Name : $9. Score1-Score3 Team ~ $25. Div $;
   datalines;
Smith,12,22,46,"Green Hornets, Atlanta",AAA
Mitchel,23,19,25,"High Volts, Portland",AAA
Jones,09,17,54,"Vulcans, Las Vegas",AA
;

proc print data=scores noobs;
run;
```

Output from Example with Format Modifiers

```
  Name      Score1    Score2    Score3             Team              Div

 Smith        12        22        46      "Green Hornets, Atlanta"    AAA
 Mitchel      23        19        25      "High Volts, Portland"      AAA
 Jones         9        17        54      "Vulcans, Las Vegas"        AA
```

CONCLUSION

In SAS language colon can be used in many different ways. The colon is a very powerful tool for maintaining clear, concise code. In particular, its prefixing power along with programming conventions can produce smaller and more easily understood code. This in turn means fewer errors in the program and less typing for programmer. The colon can be employed in managing temporary variables, formatting similar variables, dealing with unknown number of variables and cleaning up after macro execution. In short, the colon can be used across the SAS system to simplify programs and increase productivity. Mastering differently functionality of SAS colon gives you added flexibility in writing efficient functional code.

REFERENCES

1. SAS Institute, SAS® OnlineDoc, Version 9.1.3, 2004, Cary, NC:SAS Institute, Inc.
2. Luo, Haiping, SUGI 26: That Mysterious Colon (:)

3. Proskin, Howard, SUGI 29 : Colonoscopy for the SAS® Programmer

ACKNOWLEDGMENTS

I would like to thank James Gaiser for supporting me in writing the paper and Jon Seltzer for his suggestions.

CONTACT INFORMATION

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

Sudhir Singh
EMD Serono
One Technology Place
Rockland, MA 02370 USA
781-681-2565
Sudhir.singh@merckgroup.com