Effects of the KEEP, DROP and RENAME Statements and Corresponding Data Set Options
John R Whittington, Mediscience Services Ltd., Buckingham, UK

ABSTRACT
The effects KEEP, DROP and RENAME statements of the SAS® language are briefly reviewed in comparison with the corresponding KEEP=, DROP= and RENAME= data set options, including the question of whether the positioning of the statements within a DATA step is important. Performance of the data set options for subsetting variables for processing by PROCs is investigated, in comparison with the preliminary creation of reduced size data sets or data views in a DATA step, and recommendations made as regards the most execution-time-efficient approaches, particularly when the same subset of variables is to be processed by more than one PROC. The performance implications of the (unintended) anomalous behaviour of data set options applied to the input data set of PROC SORT are investigated, and recommendations made for optimising performance when sorting of variable subsets is required.

INTRODUCTION
The KEEP, DROP and RENAME statements of the SAS® Language are mirrored by the corresponding KEEP=, DROP= and RENAME= ‘data set options’ which can be applied to the input and/or output data sets in a SAS DATA or PROC step.

In virtually all DATA step situations, the same functional result can be achieved by use of either the statements or the data set options, and in some situations the same functional result can also be achieved by applying the data set options to either the input or output data sets. There has been considerable discussion on the SAS-L internet discussion group as to whether there are any differences, in terms of either function or performance) between the statements and data set options, or a difference in performance between the application of data set options to the input and output data sets when both achieve the same functional result - particularly in relation to PROC steps.

THE DOCUMENTED DIFFERENCES BETWEEN THE STATEMENTS AND DATA SET OPTIONS
The SAS Language documentation mentions the same two evident differences between the statements and data set options for all three of the functionalities under consideration - KEEP, DROP and RENAME. These two differences are:

- The statements cannot be used in SAS PROCs, whereas the data set options can.
- The statements necessarily apply to all output data sets created by the DATA step, whereas the data set options can be different for different output data sets.

In addition, in the case of RENAME and RENAME=, the documentation also mentions the evident fact that the RENAME= data set option (applied to the input data set) must be used if one wishes to rename variables prior to their being processed by the DATA step.

Since the KEEP, DROP and rename statements cannot be used in SAS PROC steps, comparison of them with data set options necessarily relates only to DATA step processing.

WHEN AND WHERE DO THEY ACT?
DROP, KEEP and RENAME statements effect only what is written from the Program Data Vector (PDV) to output data sets, and therefore are equivalent in function to application of the corresponding data set options to the output data set.

It is therefore important to realise that, even when the statements are used, variables which are to be DROPped (or not KEEPped) remain in the PDV and are available for processing within the DATA step.

Similarly, when the RENAME statement (or RENAME= data set option applied to the input data set is used, the variable retains its old name throughout DATA step processing, and is changed only when the observation is written out to the output dataset at the end of a DATA step iteration (or on encountering an OUTPUT statement). A variable with the ‘new’ name therefore does not exist for processing within the DATA step.

Conversely, when the DROP=, KEEP= or RENAME= data set options are applied to the input data set, then the intention is clearly that they take effect prior to any processing in the step. In other words, they are intended to control what goes in to the PDV and hence becomes available for processing.

DOES THE POSITION OF THE STATEMENTS WITHIN A DATA STEP MATTER?
The KEEP, DROP and RENAME statements are not ‘executable’ and therefore generally can be placed anywhere within a DATA step without altering their action. Indeed, some programmers have a policy of DROPping variables such as those used as index variables of loops immediately before, or immediately after, the first occurrence of that variable, in order to avoid that DROPping being overlooked - for example:

```sas
drop i;
do i = 1 to 10 ;
```

OR

```sas
do i = 1 to 10 ; drop i ;
```

The positioning of a RENAME statement never matters, but there is at least one situation in which positioning does matter with DROP and KEEP statements - namely when they are used with the ‘special variable lists’ _NUMERIC_ or _CHARACTER_. In such cases, just as in the case of the RETAIN statement, these special lists will only reference those variables which have been defined (by assignment, or by reading with INPUT, SET etc.) prior to the KEEP or DROP statements. Hence the following code:
data one;
  set test; /* has numeric vars n1 n2 and character vars c1 and c2 */
  n3 = n1;
  drop _numeric_;
  n4 = n2;
run;

... will result in n1, n2 and n3 being DROPped, but not n4. Care has therefore to be exercised in positioning the KEEP or DROP statements if they are used with _NUMERIC_ or _CHARACTER_. In the most common situation (where one wishes to refer to all character or all numeric variables in the DATA step), it is usually best to put such statements at the very bottom of the DATA step.

INVESTIGATING THE BEHAVIOUR OF THE STATEMENTS AND DATA SET OPTIONS

In the case of a DATA step, it is relatively easy to investigate behaviour of these programming elements, since one can actually 'look' for the presence of variables in the PDV, by using PUT statements. It is thereby possible to confirm that these programming elements all behave as described above when used in DATA steps. For example, the following code enables confirmation that KEEP= (and DROP=) data set options applied to input data sets prevent 'unwanted' variables being placed in the PDV.

```sas
data try;
  set test (keep = a c);
  put _all_;
run;

data try;
  set test (keep = _numeric_);
  put _all_;
  proc print; run;
```

In the case of PROC steps, however, where our interest is in examining the effects of data set options applied to the input and (where appropriate) output data sets, there is no such direct method, because it is not possible to 'look' directly inside the step to see what variables are present. Instead one has to rely more on examining aspects of performance - which are, in any event, what is ultimately of interest to us.

For the purpose of such investigations, a modest-size test data set was created, consisting of 500 numeric variables, with assigned random values, and 10,000 observations, and the tests carried out using SAS for Windows 6.11 (TS040) running on a 133 MHz Pentium personal computer with 40 MB RAM.

Subsetting Variables with most PROCs (excluding SORT)

The main experiments were undertaken to determine whether KEEP and DROP statements applied to the input data set do appear to prevent unwanted variables being passed to the PROC for processing. Results obtained were similar for most PROCs, and are typified by those obtained for PROC MEANS, using all its default specifications. PROC MEANS was run on all 500 variables, and then just on one variable, this being selected by one of several methods:

- KEEP= data set option on PROC MEANS input data set
- Using a DATA step to create a data set with one variable for processing by PROC MEANS
- Creating a data view with just one variable, and processing this view with PROC MEANS

The following code was run 20 times:

```sas
proc means data = test ; run ;
proc means data = test (keep = x1) ; run ;
data narrow (keep = x1) ;
  set test ;
  run ;
  proc means data = narrow ; run ;
data narrow2 (keep = x1) / view = narrow2 ;
  set test ;
  run ;
  proc means data = narrow2 ; run ;
```

... and the mean execution times for each step was determined, and are presented here graphically:

As expected, there was a dramatic difference between the execution time for the 500-variable data set and the 1-variable data set (created by a DATA step) - approximately a 100-fold difference (0.3 sec vs. 35 sec.). In contrast, when one variable was selected for processing by means of a KEEP= data set option on the input data set, the difference was far less, this process taking 17 seconds - almost exactly half of the time taken to process the full 500-variable data set. The existence of an appreciable, albeit modest (50%) reduction in execution time certainly suggests that the KEEP= data set option is actually preventing all 500 variables being passed to PROC MEANS for processing. Were all 500 variables being processed by PROC MEANS, the only potential time saving would be a reduction in the amount written to the output window - and the amount of output from PROC MEANS is far too small to account for an 18 second reduction in execution speed.

Examining the results further, it can be seen that when the time for the variable-reducing DATA step is added to the very small time taken for PROC MEANS to process one variable, the total time (19 seconds) is very close to that taken when the KEEP= (input) data set option was used with PROC MEANS itself. This therefore suggests that 18-19 seconds was taken to read the full input dataset and 'select' one variable, whether this was done with a KEEP= data set option on the PROC step or by means of a DATA step.
Finally, a data step view which contained only one variable was created from the source data set; creating that view took only about 0.4 seconds. When PROC MEANS processed this view, the time taken was again 19 seconds, virtually all of which time was in the creation of the data from the view, not processing. This is again consistent with the time taken to extract one variable from the 500 being about 18-19 seconds, regardless of how that is achieved.

Very similar results were obtained with PROCs other than PROC MEANS, and are the same whether one uses KEEP= or the corresponding DROP= option, so the following conclusions probably apply generally to situations in which one wishes to select a small number of variables from a data set containing many variables for processing by a proc:

- when considered as a one-off exercise, all three methods considered for selecting a small number of variables for processing resulted in similar execution times.
- the reduction in execution time achieved was only modest - approximately 50% when selecting one variable from 500.
- although the total time (DATA step + PROC step) to create and process ‘narrower’ data was identical whether with a reduced-size data set or with a data view, there is an appreciable performance consideration if the same ‘narrower’ set of data is to be processed by more than one PROC. In the first case (creating a data set), virtually all of the total time was accounted for by creation of a ‘narrower’ data set, not in execution of the PROC. In contrast, creation of a data view takes very little time, virtually all of the total time being taken by the subsequent PROC (and nearly all of that time taken is in creating data from the view).
- If the same small subset of variables is going to be processed through more than one PROC, creation of a ‘reduced’ data set therefore can afford a considerable execution time saving as compared with creating a data view.

Subsetting variables with PROC SORT

The one PROC which appears to behave differently (at least up to SAS release 6.11 TS040) is PROC SORT. A similar investigational process was undertaken to that described above, but in this case with one additional approach - application of the KEEP= data set option to the output data set.

The following code was run 20 times:

```
proc sort data = test out = done ; by x1 ; run ;
proc sort data = test (keep = x1) out = done ; by x1 ; run ;
data narrow (keep = x1) ;
  set test ;
  run ;
proc sort data = narrow out = done ; by x1 ; run ;
data narrow2 (keep = x1) / view = narrow2 ;
  set test ;
  run ;
proc sort data = narrow2 out = done ; by x1 ; run ;
```

... and the mean execution times for the various approaches are shown here graphically:

The KEEP= data set option, selecting one variable out of 500, produced only a very modest reduction (from 102 sec. to 84 sec.) in the execution time of PROC SORT when the option was applied to the input data set, and virtually the same reduction (approximately 20%) in execution time (to 82 sec.) when applied to the output data set.

In contrast, when a 1-variable data set was created in a DATA step, there was a very dramatic reduction in execution time for PROC SORT (approximately 0.6 seconds) - when combined with the execution time of the variable-subsetting DATA step, the total execution time was still only 21 seconds - about an 80% reduction as compared with the full 50-variable dataset.

As with the other PROCs discussed above, creation of a data view resulted in virtually the same total execution time (DATA step + PROC SORT) as when a reduced size data set was created - but, again, virtually all of this time was in the PROC SORT (actually, in creating data from the view), not in the creation of the view.

This evidence casts considerable doubt as to whether use of the KEEP= (or DROP=) data set option applied to the input data set of PROC SORT actually does prevent all variables from the input dataset being passed to the PROC and sorted. This is supported by the fact that virtually identical execution times resulted from application of the KEEP= data set option to input and output datasets, despite the fact that other methods of variable-subsetting (reduced data set or data view) resulted in a considerable reduction in execution time of PROC SORT. It is believed that this is a known design ‘bug’ in PROC SORT, the effect of which is that data set options applied to the input data set do not appear to be applied until after sorting (of the full input data set) has been completed.

The existence of this ‘bug’, applying to all data set options applied to the input data set of PROC SORT, can be confirmed by examining the RENAME= option. The following code should work, since the variable name should be changed prior to data being passed to PROC SORT for processing - so that the BY variable would be valid:

```
data test (drop = i) ;
do i = 1 to 100 ;
x1 = ranuni(1234567) ;
  output ;
extend ;
run ;
proc sort data = test out = done (rename = (x1 = y1)) ;
by y1 ;
run ;
```
However, failure of RENAME= to operate prior to processing means that variable y1 is not available to the PROC, creating an error condition as shown by the following log:

```
11 proc sort data = test out = done (rename = (x1 = y1)) ;
12 by y1 ;
ERROR: Variable Y1 not found.
14 run ;
```

However, if one sorts BY x1 (the name before re-naming), then the output data set does contain the changed name for the variable (y1). This confirms that data set options applied to the input dataset of PROC SORT act after the data have been processed - i.e. in the same manner as data set options applied to the output data set.

So long as this (presumably unintended) behaviour of PROC SORT persists, there is therefore a considerable time benefit to be obtained by subsetting the variables prior to a PROC SORT step - using a DATA step to create either by a reduced data set or a data view. As in the case of the other PROCs discussed above, if the same subset of variables is to be used as the input to more than one PROC, then creation of a reduced-size data set may be much more time efficient than creation of a data view.

CONCLUSION

The differences between the KEEP, DROP and RENAME statements and the corresponding data set options have been reviewed. Positioning of the statements in a DATA step is unimportant, unless _NUMERIC_ or _CHARACTER_ are used as 'special' variable lists. When it is desired to process a subset of variables in a PROC step, there is little to choose in terms of one-off performance between use of KEEP or DROP data set options (applied to the input data set), creation of a variable-subsetted data set or creation of a variable-subsetted data view. However, if the same subset of variables is to be processed by several PROC steps, then creation of a data set with the required subset of variables can offer a considerable performance benefit over either of the other two approaches. Data set options applied to the input data set of PROC SORT act as if they had been applied to the output data set, which is a design 'bug' present up to at least SAS release 6.11 (TS040). Until the problem is rectified, a performance advantage will usually result from creating a data set or data view with the required variables for processing by PROC SORT. The former of these two approaches (data set, not view) again being preferable if the same subset of variables is to be processed by more than one PROC step.

SAS is a registered trademark of SAS Institute Inc. in the USA and other countries. © indicates USA registration

John R Whittington
Mediscience Services Ltd.,
Twyford Manor,
Twyford, Buckingham MK18 4Ei, UK
Voice: +44 1296 730225
Fax: +44 1296 738893
e-mail: johnw@mag-net.co.uk
    100517.3677@compuserve.com