The SET Statement and Beyond: Uses and Abuses of the SET Statement

S. David Riba, JADE Tech, Inc., Clearwater, FL

ABSTRACT

The SET statement is one of the most frequently used statements in the SAS® System. It is also probably one of the more misused SAS statements because the SET statement typically works without much programmer input. Proper use of the SET statement is one of the key techniques to improving the efficiency of SAS programs.

INTRODUCTION

The function of the SET statement is to process existing SAS data sets as input for a DATA step. With no options specified, the SAS System sequentially reads each observation in the named data sets, one observation at a time, until there are no further observations to process. The SET statement is very flexible and has a variety of uses. With some planning by the SAS programmer, proper use of the SET statement and SET statement options can improve the efficiency of most SAS programs. This paper will look at the options for the SET statement and suggest some ways that they can be utilized.

SET STATEMENT OVERVIEW

The simplest form of the SET statement is

SET *dsname* ;

With no options specified, the SAS System sequentially reads each observation in the named data set(s), one observation at a time, until there are no further observations to process. If multiple data sets are listed, all observations in the first data set are read before the SAS Supervisor starts reading the second data set, etc. All variables in the original data set(s) are added to the Program Data Vector of the DATA Step. This process continues until an end of file condition occurs after reading the last observation in the last data set listed in the SET statement. To better understand the operations of the SET statement, there is a brief review of the SAS Supervisor and Program Data Vector concepts on the next page.

Up to 50 SAS data sets can be specified in a single SET statement. All the data sets must exist, although they may be empty (contain no observations). A fatal error will occur and the DATA step will terminate if a data set name that does not exist is referenced in a SET statement.

The SET statement can be used to:

- **Concatenate** multiple data sets

  SET *dsname_1 dsname_2 … dsname_n* ;

  Each data set is read sequentially from the first observation in the first named data set to the last named observation in the last data set. All variables are added to the Program Data Vector.

  The attributes of the variables are based on the first occurence of each variable. If all variables are not present in all the data sets, the values will be set to missing where the observations are from data sets without the variables. If the variables have different attributes and have not been previously defined in the DATA step, the first data set that defines a variable also defines the attribute for that variable. An error condition occurs if the TYPE of a variable differs between the data sets (ie character variable in some data sets, but numeric variable in others).

- **Interleave** Multiple SAS Data Sets

  SET *dsname_1 dsname_2 … dsname_n* ;

  BY *varlist* ;

  If the data sets are sorted, they can be interleaved based on order of the sorted variable(s). When a BY statement is used, all observations that belong to a particular BY group are read sequentially from each data set. When there are no more observations to read in any of the data sets for a particular BY group, the next BY group is selected and the process repeats itself. This continues until all observations in all the data sets have been read.

  There was a change in Version 6 of the SAS System that affects how the BY statement is applied:

  - Prior to Version 6, the BY statement was global in a DATA step. All data sets listed with any SET statement in a DATA step had to be sorted in the BY variable order. If a DATA step had multiple SET statements, the BY statement applied to all the data sets listed.

  - In Version 6, the BY statement applies only to the last encountered SET statement. Multiple SET statements now require multiple BY statements. Thus, it is now possible to have a SET statement with a BY statement and a SET statement with the POINT = option **in the same DATA step**.

- **Combine** Multiple SAS Data Sets

  SET *dsname_1* ;
  SET *dsname_2* ;

  In this example, the SAS Supervisor maintains two pointers, one for each data set. An observation would be read from data set *dsname_1* followed by an observation from data set *dsname_2*, and so on until an end of file condition occurs in one of the data sets. The combination of the two data sets would form a single observation in the new data set. Where the same variables exist in both data sets, the values of the second data set would overlay the values of the first data set.

  If (_n_ eq 1) THEN **SET** *dsname_1* ;
  SET *dsname_2* ;

  In this example, every observation in data set *dsname_2* would be merged with the first observation in data set *dsname_1*. If the first data set had grand totals, for example, these values would be combined with every observation in the second data set.

THE SAS SUPERVISOR AND THE SET STATEMENT

The Program Data Vector (PDV) is a set of buffers that are managed by the SAS Supervisor to temporarily store every variable that is referenced in a DATA step. When SAS data sets are listed in a SET statement, an allocation must be made in the Program Data Vector buffers for each variable that is defined in every data set listed.

The SAS Supervisor also controls the initialization of variables to missing between each iteration of the DATA step. In Version 6 of the SAS System, the Initialize to Missing (ITM) process has been optimized.

There are several SET statement options that affect how the SAS Supervisor manages the Program Data Vector and the Initialize to Missing process. Effective use of these options can significantly improve the efficiency of a SAS program. An understanding of how the SAS Supervisor treats the SET statement and its options is essential to the proper selection of SET statement options.

When the SAS Supervisor encounters a SET statement in a DATA step, the SAS Supervisor performs several actions:

- determines which data set to read and sets the IN = and END = variable values

- identifies the next observation to read from the data set

- assigns missing values to variables that will not be read from the current data set ***if*** the SET statement will read from a different data set than was last read

- copies the values of variables from the selected observation in the current data set to the Program Data Vector, uncompressing the input if necessary

If multiple data sets are read with a BY statement, the SAS Supervisor additionally performs the following:

- selects a data set to read by looking ahead to the values of the variables in the BY statement for the next observation

- sets the appropriate FIRST. and LAST. variable values for the BY variables

EFFICIENCY AND THE SET STATEMENT

The key to effectively using SET statement options for the highest efficiency is very simple:

Know Thy Data

Is your data:

Short and wide (few observations, many variables) ?
long and thin (many observations, few variables) ?
long and wide (many observations, many variables)?

The type of data to be processed will affect the SET statement options to use for the most efficient program. If possible, try to test the alternatives to select the most efficient option for your data.

SET STATEMENT DATA SET OPTIONS

There are several data set options that can be specified with the SET statement. These options are enclosed in parentheses and follow immediately the name of the data set that they apply to. Many of these options are also used in the DATA statement, and PROC statements such as PROC SORT and PROC PRINT.

SET *dsname_1* **(OPTION =** *list* **)**
dsname_2 **(OPTION =** *list* **)** ;

The data set options function identical to the SAS statements of the same name, with one key difference. Unlike SAS statements which operate on the variables in the Program Data Vector, SET statement options operate on variables **before** they are transferred to the Program Data Vector.

These data set options are:

DROP = *varlist*
KEEP = *varlist*
FIRSTOBS = *num*
IN = *var*
OBS = *num*
RENAME = *varlist*
WHERE = *condition*

The **DROP =** and **KEEP =** options specify which variables in the input data set are to be omitted or processed in the data step. They apply only to the data set name most recently referenced, so different DROP = or KEEP = lists can be specified when multiple data sets are listed with a single SET statement.

```
DATA newdata ;
    SET  dsname_1 dsname_2 ;
    -------
    DROP list ;
RUN ;

DATA newdata ;
    SET  dsname_1 ( drop = list )
         dsname_2 ( drop = list ) ;
    -------
RUN ;
```

Unlike the DROP statement, variables that are specified in a DROP = list are **not** available for use during a DATA step. If the same variable is specified in both a DROP = and KEEP = statement, the variable will be dropped.

For data sets that are very wide (ie data sets with many variables), considerable processing efficiencies can be achieved by dropping unnecessary variables **before** they are added to the Program Data Vector.

The **RENAME =** option renames variables exactly the same as the RENAME statement. However, since this is a SET statement data set option, the RENAME operation occurs **before** the variable is added to the Program Data Vector. Unlike the RENAME statement, your SAS code should reference the NEW name instead of the old name.

```
DATA newdata ;
    SET dsname;
    -------
    oldname = value ;
    RENAME oldname = newname ;
RUN ;

DATA newdata ;
SET dsname (rename=(oldname=newname) );
    -------
    newname = value ;
RUN ;
```

The SAS Supervisor DROPs or KEEPs variables **first** when the DROP = or KEEP = data set options are specified. It is important, therefore, to DROP or KEEP the original names and not the RENAMEd names.

The **FIRSTOBS =** and the **OBS =** data set options are often confused. Both options take a positive number as an argument. **FIRSTOBS =** specifies the observation number in a data set that is the starting observation for the SET statement. **OBS =** specifies the observation number that is the last observation in a data set to read. Thus,

```
DATA newdata ;
    SET dsname ( firstobs = 100 obs = 50 )  ;
RUN ;
```

is an impossibility and would result in a fatal error. However,

```
DATA newdata ;
    SET dsname ( firstobs = 50 obs = 100 )  ;
RUN ;
```

will sequentially read observation numbers 50 through 100 from data set *dsname*.

The combination of the FIRSTOBS = and OBS = options creates some very useful capabilities:

While debugging a SAS program, setting OBS = 0 is the same as setting the global OBS option to 0 for that DATA step -- the SAS Supervisor performs a syntax check but does not process any data. However, setting OBS = 0 as a data step option is more efficient than setting the global OBS option to 0 for that DATA step.

It would be trivial to set the OBS = variable to a macro variable which could contain the value of

0	syntax check only
x	*x* observations
the NOBS = variable	read all observations (more on NOBS later)

By changing the value of the macro variable that OBS = uses, it is possible to change a DATA step from test mode to production mode without changing any code.

In the following example, a data set is combined with itself to calculate the change in a variable's value

```
DATA newdata ;
 SET dsname_1
    ( keep = total  rename = ( total=oldtot ) )  ;
 SET dsname_1 ( firstobs = 2 ) ;
  delta = total - oldtot ;
RUN ;
```

The SAS Supervisor maintains two read pointers into the same data set, one for each SET statement. For each pass of the DATA step, an observation is read from data set *dsname_1* and then the next observation is read from the same data set. The RENAME = option is important to prevent the variable values from being replaced when the same variables are read with the second SET statement. The end of file condition will occur when the second SET statement reads the last observation in the data set. In effect, this example is an offset one to one merge with the same data set.

Use of the FIRSTOBS = or OBS = options can be a very efficient method of selecting a subset of observations in a data set. If your program selects a subset of data, for example the first 100 records or the last 100 records, consider using FIRSTOBS = or OBS = instead of reading in all the records in a data set and then deleting the unwanted records.

Note that the FIRSTOBS = and OBS = options can not be used with a WHERE statement or the WHERE = data set option. The FIRSTOBS = and OBS = options can be used with compressed data sets, but they function more slowly than if the data sets were not compressed.

The **IN =** data set option is used with multiple data sets where it is important to know which data set contributed an observation. A separate IN = variable can be specified for each data set defined with the SET statement. The variable named by the IN = option has a value that is set to 1 for every observation that originated from the data set.

```
DATA newdata ;
   SET dsname_1 ( in = in_1 )
       dsname_2 ( in = in_2 ) ;
   -------
   IF ( in_1 ) THEN ----------- ;
   ELSE IF ( in_2 ) THEN ----------- ;
RUN ;
```

The variables defined with the IN = option only exist for the duration of the DATA step and are not added to the data set. Do not DROP or KEEP these variables.

The **WHERE =** data set option selects observations from a SAS data set that meet the conditions specified. It functions identical to the WHERE statement. However, the WHERE = data set option is more efficient than the WHERE statement, because only those observations that match the conditional test are transferred into the Program Data Vector. With the WHERE statement, all observations are read from the input data set and non-selected observations are discarded. The WHERE = option only selects those observations that match the criteria.

```
DATA newdata ;
   SET dsname_1 ( where = ( condition ) )
       dsname_2  ( where = ( condition ) ) ;
 -------
RUN ;
```

If both a WHERE = and a WHERE statement are used in the same data step, the WHERE statement is ignored for those data sets that have a WHERE = condition defined.

The WHERE = data set option can not be used with the POINT =, FIRSTOBS = , or OBS = data set options.

SET STATEMENT OPTIONS

Besides the data set options, there are several other options that are also used with the SET statement. With the exception of the newly available KEY = option, each of these options uses a SAS variable that is valid during the DATA step. However, these variables are **not** added to the final data set. Unlike the SET statement data set options, these options are not enclosed in parentheses.

These SET statement options are:

END = *var*
KEY = *index*
NOBS = *var*
POINT = *var*

The **END =** option is used to identify the last observation processed by a SET statement. It creates a variable whose value is set to 0 for overy observation except for the last observation in the last data set (end of file of the final data set). When the last observation is read, the variable value is set to 1. DATA step logic can now be used to perform specific tasks before the DATA step terminates.

```
DATA newdata ;
   SET dsname_1 dsname_2 end = eof ;
   -------
   IF ( eof ) THEN DO ;
      ------
      ------
   END ;
RUN ;
```

Since the variable created by the END = option is not added to the final data set, it would be an error to try to DROP or KEEP the variable.

The **KEY =** option retrieves observations from an indexed data set based on the index key, which can be either a simple key or a composite key. If no observations are found in the data set that match the value of the key variable(s), then an error condition occurs.

```
      DATA newdata ;
         SET *dsname_1 ;*
         SET *dsname_2* **key =** *indexvar* ;
      RUN ;
```

The **NOBS =** option creates a variable which contains the total number of observations in the input data set(s). If multiple data sets are listed in the SET statement, the value in the NOBS = variable are the total number of observations in all the listed data sets.

```
      DATA newdata ;
         SET dsname_1 dsname_2 nobs = totobs ;
      RUN ;
```

Of key importance is that the value of the NOBS = variable is set by the SAS Supervisor at compile time. During the compile phase of a DATA step, the SAS Supervisor retrieves the number of observations in each data set listed in a SET statement from the header record of the data set. When the SAS System begins to execute the DATA step, the value for this variable is already defined and can be referenced immediately. Because the value of the NOBS = variable is already defined when the DATA step starts executing, it can be used **before** the SET statement is acted upon.

The following example looks impossible, but is actually a very useful application of the NOBS = option:

```
 IF (0) then SET dsname ( drop=_ALL_ ) nobs=totobs;
```

This statement assigns the total number of observations in data set *dsname* to a variable called ***totobs***, which can be used anywhere in the DATA step. It is never executed, since the IF (0) condition can never be True. Also, the variables in data set *dsname* are never added to the Program Data Vector because of the DROP = _ALL_ option on the SET statement. The purpose of this statement is to store the number of observations in a data set in a variable called TOTOBS without adding the observations or variables in the data set to the Program Data Vector of the current DATA step. This might be useful in preventing a data set that has no observations from being used as input to update an existing data set.

This example could be used with the following statement:

```
      IF ( totobs ) then SET dsname ;
         ELSE abort ;
```

To define the number of observations in a data set in a macro variable (&N_OBS) for later use in a SAS program:

```
      DATA _NULL_ ;
      CALL SYMPUT ( 'n_obs' , put ( n_obs, 5. ) ) ;
       STOP ;
         SET dsname nobs = n_obs;
      RUN ;
```

The only executable statement in this DATA step is the CALL SYMPUT to create macro variable &N_OBS. The SET statement is never executed, but the value of variable *n_obs* is assigned at compile time.

The **POINT =** option uses a numeric variable for direct (or random) access into a SAS data set. The value of the POINT = variable must be specified before it can be used.

```
      DATA newdata ;
         ptr = 100 ;
         SET dsname point = ptr ;
         IF ( _error_ ) THEN abort ;
             -------
         STOP ;
      RUN ;
```

Since any number can be specified as the value for the POINT = variable, it is important to verify that the number is a valid observation number (eg compare it to the variable created by the NOBS = option). If the pointer tries to access an observation number that is greater than the number of observations in the data set, an error condition occurs.

The variable used for the POINT = option (the pointer) is temporary and is not added to the final data set. Do not try to DROP or KEEP the pointer variable. Also, do not use a variable that already exists in any of the data sets listed with the SET statement or some very unpredictable results may occur.

If multiple data sets are listed with a single SET statement, the POINT = variable retrieves observations from the data sets as if they were concatenated. That is, the observation number between one and the last record of the last data set in the SET statement list.

The POINT = option is not valid for compressed SAS data sets, transport files, SAS/ACCESS views, or SQL views that read data from external files. It is **very important** to be aware that since the POINT = option provides direct access to SAS data sets, it can **not** be used with a BY statement. It also can not be used with a WHERE statement or the WHERE = data set option. It is also important to be aware that continuous loops can occur since there is not always an end of file indicator when using the POINT = option. Provisions should be made to end the DATA step to prevent a continuous loop.

```
      DATA newdata ;
       DO ptr = lastrec to 1 by -1 ;
          SET dsname_1 point = ptr nobs = lastrec ;
          IF ( _error_ ) THEN abort ;
          OUTPUT ;
       END ;
       STOP ;
      RUN ;
```

This example reads a data set in reverse order, last observation to first observation.

Another use of the POINT = option is to generate a random sample of data. To select 100 random observations from a data set:

```
DATA newdata ;
  DO _I_ = 1 to 100 ;
     ptr = ceil ( totobs * ranuni ( totobs ) ) ;
     SET dsname_1 point = ptr nobs = totobs ;
            IF ( _error_ ) THEN abort ;
     OUTPUT ;
  END ;
  STOP ;
RUN ;
```

The source of the pointer values can also be from another SAS data set. In this example, data set *dsname_1* contains numeric variables START and STOP which define the observation numbers in data set *dsname_2* that are to be read with the second SET statement.

```
DATA newdata ;
     SET dsname_1 ;
     DO ptr = start to stop ;
            SET dsname_2 point = ptr ;
            IF ( _error_ ) THEN abort ;
         -------
         OUTPUT ;
     END ;
RUN ;
```

It is important to be aware that direct access with the POINT = option is generally less efficient for large data sets. It is most efficient for small data sets or when reading up to a maximum of 20-25% of a data set.

DO LOOPS AND THE SET STATEMENT

Placing the SET statement inside a DO loop is usually more efficient than letting the SAS Supervisor control processing especially for large data sets.

```
DATA newdata ;
  DO UNTIL (lastrec) ;
     SET dsname_1 dsname_2 end = lastrec ;
         ---------
         OUTPUT ;
   END ;
   STOP ;
RUN ;
```

Surprisingly, this example is generally more efficient than a simple SET statement because some operations of the SAS Supervisor are bypassed. The efficiencies that can be achieved by bypassing the SAS Supervisor have some trade-off costs. Bypassing the SAS Supervisor means that variables are no longer reset to missing before each observation is read from the input data sets. Since variables are not reset to missing, the SAS programmer needs to perform this task if necessary. Also, there is no implied output performed, so it is important to specify when to add data to the new data set with an OUTPUT statement. Finally, it is very important to define some condition when to stop processing the DATA step. Note the STOP statement at the end of the DATA step.

Note that some changes to optimize Version 6 of the SAS System have decreased the efficiency gains achieved by putting the SET statement in a DO UNTIL loop. It would be advisable to test your proposed code on your current level of SAS software before implementing this type of example.

The following example calculates a minimum, maximum, and sum for a variable in a data set, for use with every observation in that data set **in the same DATA step**.

```
DATA newdata ;
    RETAIN minval maxval  sumval ;
    IF ( _N_ eq 1 ) THEN DO UNTIL (lastrec) ;
       SET dsname_1 (keep = value) end = lastrec;
       minval = min ( minval, value ) ;
       maxval = max ( maxval, value ) ;
       sumval = sum ( sumval, value ) ;
    END ;
    SET dsname_1  ;
     ---------------------
 RUN ;
```

CONCLUSION

This paper examined each of the options for the SET statement, considering the efficiency issues that applied to each option. Options such as DROP = and KEEP = can impact the processing of data sets with many variables. Options such as FIRSTOBS =, OBS =, WHERE =, and POINT = can select the observations to read before they are copied to the Program Data Vector by the SAS Supervisor. These options can be combined for the highest efficiency. The key is an understanding of the data to be processed and how the SAS Supervisor interacts with the SET statement.

A proper understanding of the SET statement and the options that can modify it can have a significant impact on the efficiency of the DATA step. When reading SAS data sets, the SET statement can control which variables to include in the Program Data Vector, which observations to read from a SAS data set, or both. Proper use of the SET statement and SET statement options can enhance program efficiency, while improper use can reduce DATA step efficiency.

REFERENCES

Howard, N. (1991). Efficiency Techniques for Improving I/O and Processing Time in the Data Step. *Proceedings of the Sixteenth Annual SAS Users Group International Conference*. SAS Institute, Inc., Cary, NC. pp. 284-289.

Rabb, M. G., Henderson, D. J., and Polzin, J. A. (1992). The SAS System Supervisor - A Version 6 Update. *Proceedings of the Seventeenth Annual SAS Users Group International Conference*. SAS Institute, Inc., Cary, NC. pp. 190-197.

SAS Institute, Inc., *SAS® Language: Reference, Version 6, First Edition* Cary, NC. SAS Institute Inc., 1990.

Schallert, Bill (1992). Use of the KEEP and DROP Concept in Batch SAS® Programs. *Proceedings of the Seventeenth Annual SAS Users Group International Conference*. SAS Institute, Inc., Cary, NC. pp. 190-197.

Thornton, R. G. and Rabb, M. G. (1988). Advanced Set and Merge Processing. *Proceedings of the Thirteenth Annual SAS Users Group International Conference*. SAS Institute, Inc., Cary, NC. pp. 1168-1177.

Zeid, Aiman. (1989). The SAS Supervisor and SET / MERGE Processing. *Proceedings of the Fourteenth Annual SAS Users Group International Conference*. SAS Institute, Inc., Cary, NC. pp. 17-26.

ACKNOWLEDGEMENTS

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

AUTHOR

The author may be contacted at:

S. David Riba
JADE Tech, Inc.
P O Box 4517
Clearwater, FL 33758
(813) 726-6099
INTERNET: dave@JADETEK.COM

This paper is available for download from the author's Web site:

HTTP://WWW.JADETEK.COM/JADETECH.HTM

SPEAKER BIO:

S. David Riba is CEO of JADE Tech, Inc., a SAS Institute Quality Partner who specializes entirely in applications development and training in the SAS® System.

Dave is the founder and President of the Florida Gulf Coast SAS Users Group. He chartered and served as Co-Chair of the first SouthEast SAS Users Group conference, SESUG '93, and serves on the Executive Boards of both the SouthEast SAS Users Group and the Consultants SAS Users Group (CONSUG). His first SUGI was in 1983, and he has been actively involved in both SUGI and the Regional SAS Users Group community since then. He has presented papers and assisted in various capacities at SUGI, SESUG, NESUG, MWSUG, SCSUG, and PharmaSUG.

Dave is an unrepentant SAS bigot. His major areas of interest are efficient programming techniques and applications development using the SAS® System, particularly using Screen Control Language with SAS/AF® and FRAME technology.