

# The DATA Statement: Efficiency Techniques

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

## ABSTRACT

One of those SAS<sup>®</sup> statements that everyone learns in the first day of class, the DATA statement rarely gets a second look. However, this basic statement has evolved in recent releases of the SAS System. If you have not looked at the options on the DATA statement recently, you will probably be surprised.

This Beginning Tutorial examines the syntax of the DATA statement and the different options that are valid for use with it. Each option will be discussed and illustrated with examples of how (and when) to use them. Among the topics that will be discussed are dataset indexing, compression, and views.

## EFFICIENCY AND THE DATA STATEMENT

There are many options that can be used to control the default behaviors of the DATA statement. The proper use of these options can be very effective in improving the efficiency of your SAS programs. Among these are options to:

- Manage Dataset Variables
- Manage Datasets
- Control Access to Data
- Define Indexes and Views

There are additional options to the DATA statement that will not be included in this Tutorial. These include options to manage host system processing, such as BUFNO, BUFSIZE, CNTLLEV, or TYPE. Those options are less frequently used, and are not particularly useful in the context of this Tutorial.

In addition to the DATA statement options, this tutorial will examine the Data Step Debugger and the Stored Program Facility. Both of these optional parameters to the DATA statement can be used very effectively to improve the efficient processing of your programs. The Data Step Debugger can be used for interactive debugging of the data step. Bottlenecks, dead-end code, and many other code problems can be isolated and fixed through the proper use of the Data Step Debugger. The Stored Program Facility can be used to pre-compile data steps, so they do not have to be compiled every time they are executed.

## DATA STATEMENT OVERVIEW

The simplest form of the DATA statement is:

```
DATA dsname ;
```

With no options specified, the DATA keyword causes two primary actions to occur. The DATA keyword is used to signal to the SAS System that a new step is about to begin. When a DATA keyword is encountered, SAS promptly completes processing on any previous step, either DATA or PROC. It then sets up the environment necessary to process the program data step that follows.

When a data step begins, the Program Data Vector is defined, various buffers are established as necessary, and at least one file is opened to store the results of the processing defined in the datastep. If no name is defined for this file, SAS uses the default name of WORKn (WORK1, WORK2, ... WORKn). If a name is defined after the DATA keyword, the file is assigned the name that was defined. If a file already exists with this name, the file is overwritten and a new file is created.

The default behavior is to store the SAS dataset files in the SASWORK library. If a two-level name is defined (i.e. SASUSER.dsname), then the file is given the name which was defined in the second part of the name (dsname) and stored in the library defined in the first part of the name (SASUSER).

The basic syntax of the DATA statement can define up to 50 SAS datasets in one invocation:

```
DATA dsname_1 dsname_2 ... dsname_n ;
```

## SYNTAX FOR DATASET OPTIONS

Dataset options take two forms, those that modify the actions of individual datasets and those that modify the operation of the DATA statement itself.

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

The general syntax for these dataset options is:

```
DATA dsname_1 ( OPTION = list_1 )
  dsname_2 ( OPTION = list_2 ) ;
```

Note that the options can differ for each dataset.

In addition, there are options that modify the default processing of the DATA statement. These options are added at the end of the DATA statement, before the semicolon. A slash is used to separate these options from the dataset names and dataset options. These options are not enclosed in parentheses.

The general syntax for these options is:

```
DATA dsname_1 dsname_2
  / OPTION = list ;
```

Both types of options can be combined in a single data step as necessary:

```
DATA dsname_1 ( OPTION = list_1 )
  dsname_2 ( OPTION = list_2 )
  / OPTION = list ;
```

## OPTIONS TO MANAGE VARIABLES

Many of the dataset options to manage variables 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, DATA statement options operate on variables after they are transferred from the Program Data Vector to the SAS dataset.

These dataset options include:

- **DROP** = varlist
- **KEEP** = varlist
- **RENAME** = varlist

The **DROP** = and **KEEP** = options specify which variables in the Program Data Vector are to be saved in the output dataset. They apply only to the dataset name most recently referenced, so different **DROP** = or **KEEP** = variable lists can be specified when multiple datasets are listed with a single DATA statement.

This example drops variables from both datasets:

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

This example shows how the **DROP** = can be used to specify different variables for each dataset.

```
DATA dsname_1 ( drop = list_1 )
  dsname_2 ( drop = list_2 ) ;
  -----
RUN ;
```

They can be combined as follows:

```
DATA dsname_1 ( drop = list_1 )
  dsname_2 ( drop = list_2 ) ;
  -----
  DROP list ;
RUN ;
```

If the same variable is specified in both a **DROP** = and **KEEP** = statement, the variable will be dropped.

The **RENAME** = option renames variables exactly the same as the **RENAME** statement.

```
DATA dsname_1 ;
  -----
  oldname = value ;
  RENAME oldname = newname ;
RUN ;
```

```
DATA dsname_1
  (rename=(oldname=newname)) ;
  -----
  oldname = value ;
RUN ;
```

The SAS Supervisor **DROP**s or **KEEP**s variables **first** when the **DROP** = or **KEEP** = dataset options are specified. It is important, therefore, to **KEEP** the original names and not the **RENAMED** names.

The primary difference between the **DROP**, **KEEP**, and **RENAME** statements and the **DROP**=, **KEEP**=, and **RENAME**= dataset options is the scope of execution. The former are global and act on the values in the Program Data Vector. The latter are specific to a particular dataset, and may be different for each dataset named after the DATA statement.

## OPTIONS TO MANAGE DATASETS

Besides the dataset options mentioned above, there are several other options that are used with the DATA statement to manage datasets.

These DATA statement options are:

- **COMPRESS** =
- **REUSE** =

- \$POINTOBS =
- REPLACE =
- LABEL =
- SORTEDBY =

## Dataset Compression

The **COMPRESS** = option changes the default storage structure of a SAS dataset. Previously, all SAS datasets were stored as fixed length record files. This meant that every record in a SAS dataset occupied the same amount of storage space, regardless if there was any data in the record. Beginning in SAS Release 6.07, however, users were able to define datasets stored with variable length records. This meant that the storage space differed for each record, dependent upon the length of the data in each record. A SAS system option, **COMPRESS**, was added to control the default storage behavior of SAS globally. The primary disadvantage to compressed datasets was that it is no longer possible to directly access a specific record in a dataset. The **POINT** = option on the SET statement does not work for compressed SAS datasets. This will be changed with SAS Version 7.

While the **COMPRESS** system option functions globally for all datasets created in a SAS session, the **COMPRESS** = option acts directly on the dataset just named. Thus, some datasets can be compressed, and others can be left as fixed length records. The only valid arguments for the **COMPRESS** = option are **YES** or **NO**.

The syntax for turning on dataset compression is:

```
DATA dsname_1 (compress = YES );
-----
RUN ;
```

Note that for compressed datasets, SAS automatically reclaims space left by deleted observations. Uncompressed datasets remain the same size, even if records have been deleted.

There is another option that modifies the default behavior of compressed datasets. By default, any observations added to a SAS dataset are appended at the end of the dataset. However, empty space within a dataset due to the deletion of records could be used for new observations if the **REUSE** option were specified for the dataset.

The syntax for enabling compressed datasets to reuse deleted space is:

```
DATA dsname_1 (compress = YES
               reuse = YES );
-----
RUN ;
```

If there is sufficient space within a dataset to store the new record, it will be inserted in the middle of the dataset instead of appended to the dataset.

A new option is being added in Version 7 to allow users to access records in compressed datasets by observation number. Previously, the **POINT** = option on the SET statement fails when used with compressed datasets. The **\$POINTOBS** option on a compressed dataset determines whether or not direct access will be allowed for the dataset. The only valid values for **\$POINTOBS** are **YES** or **NO**. The default value is YES.

The syntax for \$pointobs option on compressed datasets is:

```
DATA dsname_1 (compress = YES
               $pointobs = YES );
-----
RUN ;
```

## Other Dataset Options

The **REPLACE** = option controls whether or not a dataset can be overwritten when another dataset is created with the same name. One of the major drawbacks to the DATA statement has been that a new dataset is always created with the name given after the DATA keyword. If a dataset with that name already exists, it is automatically destroyed along with any data it might include.

The global options **REPLACE** and **NOREPLACE** and the **REPLACE** = dataset option enable the user to control whether or not this default action should be allowed to occur. The default value for **REPLACE** is YES. To prevent the automatic replacement of a dataset, change the value to NO.

```
DATA dsname_1 (replace = NO );
-----
RUN ;
```

The **REPLACE** = option is only valid during a dataset. If a dataset with the given name already exists, the dataset will not overwrite it. However, a dataset created with the **NOREPLACE** option can still be modified or overwritten by a PROC step.

For example:

```
DATA dsname_1 (replace = NO )
    dsname_2 (replace = NO ) ;
-----
RUN ;
PROC SORT data = dsname_1
    out = dsname_2 ;
BY varlist
RUN ;
```

If dsname\_2 were truly stored as a non-replaceable dataset, then the output of PROC SORT should have failed. However, this code works since the procedure does not check the replacement status flag of the dataset.

The **LABEL =** option assigns a label to a SAS dataset. The label is displayed by procedures such as PROC CONTENTS and PROC DATASETS. While adding a label does nothing to improve efficiency, it can prevent confusion especially if datasets are given cryptic names.

The syntax for defining a dataset label is:

```
DATA dsname_1 (label = 'dataset label') ;
-----
RUN ;
```

The **SORTEDBY =** option is used to define variables which are already ordered in a dataset. This eliminates the need for a PROC SORT to order the variables. This is most useful if the input data is already in the proper sort order. If so, there is no need for a PROC SORT to order the data again so a BY statement can be used in PROC or DATA steps.

```
DATA dsname_1 (sortedby = var1 var2) ;
-----
RUN ;
```

## DATASET ACCESS CONTROL OPTIONS

There are several options to control access to the data in SAS datasets. Datasets can be password protected and they can be encrypted. There are four levels of password protection that can be defined.

- **PW =** defines a read and write password for a dataset. Any attempts to open the dataset will require the user to enter the correct password.
- **READ =** and **WRITE =** defines read passwords and write passwords, respectively. Note that since each can be defined independently, a dataset can have different read and write passwords.

- **ALTER =** defines a password for use when a dataset is changed.

The syntax for assigning a dataset password is:

```
DATA dsname_1 (pw = password) ;
-----
RUN ;
```

Any passwords that are assigned in the program are not listed in the SAS log when a program is executed. Instead, the SAS log contains a string of X's up to the length of the password. Thus, the SAS log would typically show:

```
DATA dsname_1 (pw = XXXXXXXX) ;
-----
RUN ;
```

The **PWREQ =** option is being introduced in SAS Version 7. It controls whether or not a Password Requester window pops up when a password protected dataset is accessed. The default value of **PWREQ** is YES. A value of NO prevents the Password Requester window from popping up.

The syntax for PWREQ is:

```
DATA dsname_1 (pwreq = NO) ;
-----
RUN ;
```

This option is most useful in SCL applications, where the user is shielded from the password.

The final type of protection available to SAS datasets is encryption. This was introduced in Release 6.11 of the SAS System. Passwords only protect datasets from within the SAS System. However, anyone with access to the files outside of SAS can read the contents of SAS datasets with a bit editor or other operating system level programs. To prevent this, SAS now allows sensitive data to be encrypted.

The **ENCRYPT =** option causes the dataset to be encrypted when stored. **ENCRYPT** takes a toggle value, either **YES** or **NO**. In order to properly encrypt a file, SAS requires a password be supplied. This can be either a **PW =** or a **READ =** password. The password provides the encryption key.

A word of caution is in order. If you forget the password, it is highly unlikely that you will be able to decrypt the data in the file without significant help from SAS Institute. Additionally, the password can not be changed on an encrypted SAS dataset without recreating the dataset.

The syntax to encrypt a SAS dataset is:

```
DATA dsname_1 (pw = password
               encrypt = yes ) ;
-----
RUN ;
```

It is important to note that encryption is only valid on SAS datasets, and is not appropriate for SAS Views or Stored Programs.

## SAS DATASET INDEXES

Indexes can be added to SAS datasets. They are used by various SAS procedures to improve the efficiency of data retrieval operations.

Imagine how difficult it would be to retrieve information from a book if there were no index to point to where the target information is located. Indexed SAS datasets function similarly. An Index is a small table that is attached to a dataset which points to the records in the datasets that have specific values for the indexed variables. For example, a dataset with an index on the variable STATE would have a lookup table which points to every record where the value for STATE is Florida, every record where the value for STATE is Georgia, etc. It would be much more efficient to find those records by looking at the index table than to sequentially read every record in the dataset.

Indexes can be simple or composite, and more than one index can be assigned to a dataset. A simple index is based on a single variable, such as STATE. A composite index is based on a combination of variables, such as CITY and STATE.

The syntax for defining an index is:

```
DATA dsname_1 ( index = list )
          dsname_2 ( index = (name= list ) ) ;
-----
RUN ;
```

The definition for a simple index (e.g. STATE) lists the names of every variable to index. The definition for a composite index (e.g. CITY and STATE) must be assigned a name for the composite index. For example, CITYST could be the name for an index based on the values of CITY and STATE.

The following examples use variables named CITY and STATE to create several different kinds of indexes.

```
DATA dsname_1 (index=(state));
               /* simple index for one variable */
DATA dsname_1 (index=(city state));
               /* simple indexes for two variables */
DATA dsname_1 (index=(cityst=(city state)));
               /* composite index */
DATA dsname_1 (index=(city cityst=(city state)));
               /* simple and composite */
```

## SAS DATASET VIEWS

In the past there have been several excellent tutorials and papers on SAS dataset views. Therefore, this Tutorial will not explore Views in any significant detail.

SAS datasets are comprised of several main components. Every dataset contains a header record which includes information about the dataset, the variables in the dataset, and other key information. PROC CONTENTS can be used to display the information in a SAS dataset header record. Second, a SAS dataset may or may not contain actual data (i.e. a dataset with zero records contains no actual data). Finally, if there is an index attached to a dataset, the index is appended.

The simplest explanation of a dataset View is that it is a dataset without data. The view contains the information about the dataset in the header record, and the instructions for how to obtain and process the data to be included, but it contains no actual data. A view does not retrieve and process data until the point in the program that it is actually used. Thus, you can PROC PRINT a view. Even if the view was defined at the beginning of the program, and the PROC PRINT is executed at the end of the program, SAS does not retrieve and process the data until it is needed by the PROC PRINT. Alternatively, a view can be defined and stored permanently, if it is given a two-level name (i.e. SASUSER.dsname). The view is stored in the library defined by the first part of the name. Every time the view is referenced (by a SAS PROC, for example), the data is retrieved and processed at the point in the program that it is needed.

Views are extremely useful in a variety of situations. SAS datasets, by their nature, contain a copy of data. If an external file is read into a SAS dataset, then there is one copy of the data in the external file and another copy in the SAS dataset. Additionally, since this was done at a certain point in time, the information in the SAS dataset is frozen at that point in time. If the external file changes, the information can be different between the file and the dataset.

However, a SAS view does not contain any data. Therefore, it does not require additional storage space and it is current as of the moment that it is actually used.

The syntax to define a SAS view is:

```
DATA dsname_1 dsname_2 dsname_3
  / view = dsname_1 ;
-----
RUN ;
```

Even though a DATA keyword can define up to 50 SAS datasets, only one SAS data view can be defined in a data step. The view name must be the same as the name of one of the datasets defined after the DATA keyword. This view will be stored with the compiled instructions from the dataset. These instructions will be acted upon when the view is used, either in another DATA step or in a PROC step.

## THE STORED PROGRAM FACILITY

One of the fundamental concepts of how SAS operates is that every time you run a SAS program each step is first Compiled and then Executed. Regardless of how many times you run a SAS program, SAS first Compiles and then Executes every step sequentially. For most programs, this is not a significant waste of resources. However, for programs that have very involved data steps, the need to first compile the step every time it is run can have a major impact on system resources. Assuming that a program compiles successfully and that no changes are made between different runs of the program, it would be useful to eliminate the Compile phase every time the program runs.

Starting in Release 6.11, SAS has provided an option to eliminate the need to compile data steps every time a program runs. It is called the Stored Program Facility. This is somewhat of a misnomer, as it is only valid for the DATA step and not PROC steps. The Stored Dataset Facility might be a more accurate name. Basically, the Stored Program Facility allows data steps to be compiled (but not executed) and stored for reuse as needed. When the compiled data step is referenced in a SAS program, it is not recompiled. Certain variable information, like dataset and file names, can be defined at the time of execution.

The syntax is slightly different between creating / storing a data step and executing a stored step. To compile and store a step, use a slash to separate the DATA keyword and dataset name from the PGM option and name of the compiled program step.

```
DATA dsname_1
  / pgm = libname.dsname ;
-----
RUN ;
```

With this syntax, SAS will compile the data step and will store the compiled program step in the location specified by the **PGM =** option. It is recommended that stored program steps should be saved in a permanent library. Temporary libraries will be deleted when the current SAS session terminates. Thus, it would be necessary to recompile the stored program every time a SAS session begins.

Once a program step has been, it can be used by listing the name of the stored program step after the DATA keyword, using the **PGM =** option :

```
DATA pgm = libname.dsname ;
-----
RUN ;
```

There are several parameters that can be defined to replace values which were in effect at the time the data step was compiled. INPUT, OUTPUT, and FILENAMEs can all be redirected. Use LIBNAME and FILENAME statements instead of directly referencing the locations. Thus, these can be replaced without the need to change the program.

For example:

```
LIBNAME in 'input-library' ;
LIBNAME out 'out-library' ;
LIBNAME stored 'save-library' ;

DATA out.dsname
  / pgm = stored.progname ;
  SET in.dsname ;
-----
RUN ;
```

The dataset OUT.DSNAME is not actually created until the stored program is run. In order to execute this program, the syntax is:

```
LIBNAME in 'new-input-library' ;
LIBNAME out 'new-out-library' ;
LIBNAME stored 'new-save-library' ;

DATA pgm = stored.progname ;
RUN ;
```

Note that the LIBNAME statement will resolve to the current values at the time of execution. A stored program does not need to be repeated, since it has already been compiled and saved.

It is important to be aware that any Macro variables referenced in the step resolve when the data step is compiled. Thus, current values of any macros are not resolved at the time of execution.

In addition, both INPUT and OUTPUT can be redirected at the time of execution. This is done with the REDIRECT statement. Thus, even though the original program might have pointed to a specific input dataset, a different dataset can be specified at the time of execution. The same is true for the output results of the program.

```
DATA pgm = libname.dsname ;  
  REDIRECT INPUT oldname=newname;  
RUN ;
```

Passwords can be assigned to the datasets created by the Stored Program Facility and will be stored with the resulting SAS datasets.

## THE DATA STEP DEBUGGER

Until Release 6.11 of the SAS System, there have been few options to test and resolve data step problems. Similar to PROC steps, the data step has largely been a “black box” where code was acted upon and results derived. The user has had limited abilities to actually step through the code being processed.

This changed with the introduction of the Data Step Debugger in Release 6.11. The debugger is an interactive debugging tool that gives the user the ability to step through the code in a data step and control its execution. Like its name implies, the debugger is only valid for DATA steps.

To invoke the Data Step Debugger, use the DEBUG parameter following a slash, as follows:

```
DATA dsname_1  
  / DEBUG ;  
  -----  
RUN ;
```

When a data step is compiled with the **DEBUG** parameter, the user is presented with two windows.

- SOURCE Window to display the data step source code and highlight the currently executing line of code
- LOG Window is the interactive window. There is a command line for entering debugger commands, and the results are displayed in the window

Once the debugger is running, the user has control over the execution of the data step. Among the actions that can be performed when the debugger is invoked:

- Program instructions can be executed one line at a time, and the values of variables can be evaluated at any time
- Values of individual variables can be changed
- Sections of program instructions can be skipped
- Sections of program instructions can be rerun
- Program logic can be traced and evaluated

There are several categories of debugger commands. There are debugger commands to:

- Control Program Execution  
**GO JUMP STEP**
- Manipulate Data Step Variables  
**CALCULATE DESCRIBE EXAMINE SET**
- Manipulate Debugger Requests  
**BREAK DELETE LIST TRACE WATCH**
- Tailor the Debugger  
**ENTER**
- Terminate the Debugger  
**QUIT**
- Control the Debugger Windows  
**HELP SWAP**

With the exception of HELP and SWAP, all of the debugger commands are issued from the Data Step Debugger command line. This is different from the SAS command line. Only HELP and SWAP are issued as SAS commands. All debugger commands can be shortened to either the first letter or the first four letters of the command. For example, the JUMP command can be used to change the next program line to be executed. To make Line 10 of the step the next line to execute, issue the command **J 10** from the debugger command line.

Debugger commands can be issued from Function Keys. Multiple commands can be combined into a single Function Key, provided the individual commands are separated by a semi-colon. For a full description of the Data Step Debugger and debugger commands, refer to the SAS online documentation.

## CONCLUSION

This paper examined many of the options for the DATA statement, considering the efficiency issues that applied to these options. A proper understanding of the DATA statement and how the options can be used to modify the default behavior of the DATA statement can be very useful in improving the efficiency of SAS programs.

This Tutorial reviewed the syntax of the DATA statement. It discussed the DROP, KEEP, and RENAME options to manage dataset variables. Datasets can be managed with the COMPRESS, REUSE, \$POINTOBS, REPLACE, LABEL, and SORTEDBY options. Additionally, Indexes and SAS Data Views were examined. Finally, the Tutorial reviewed the Stored Program Facility and the Data Step Debugger.

## REFERENCES

SAS Institute, Inc., SAS<sup>®</sup> Language: Reference, Version 6, First Edition Cary, NC. SAS Institute Inc., 1990.

SAS Institute, Inc., SAS<sup>®</sup> Technical Report P-222, Changes and Enhancements to Base SAS<sup>®</sup> Software, Release 6.07, Cary, NC. SAS Institute Inc., 1991.

SAS Institute, Inc., SAS<sup>®</sup> Online Doc, V7 Beta, Cary, NC. SAS Institute Inc., 1998.

## ACKNOWLEDGMENTS

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

<sup>®</sup> indicates USA registration.



## AUTHOR

The author may be contacted at:

S. David Riba  
**JADE Tech, Inc.**

P O Box 4517  
Clearwater, FL 33758  
(727) 726-6099

INTERNET: [dave@JADETEK.COM](mailto:dave@JADETEK.COM)

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

[HTTP://WWW.JADETEK.COM](http://WWW.JADETEK.COM)

## SPEAKER BIO:

S. David Riba is CEO of JADE Tech, Inc., a SAS Institute Quality Partner who specializes entirely in applications development, consulting 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. Dave currently serves as President of the SouthEast SAS Users Group, and serves on the Executive Council of CONSUG, the Consultant's SAS Users Group. His first SUGI was in 1983, and he has been actively involved in both SUGI and the Regional SAS User Group community since then. He has presented papers and assisted in various capacities at SUGI, SESUG, NESUG, MWSUG, SCSUG, and PharmaSUG.

Dave is an unrepentent SAS bigot. His major areas of interest are efficient programming techniques and applications development using the SAS System. His SAS software product specialties are SAS/AF and FRAME technology, SAS/EIS, SAS/IntrNet, webAF, and CFO/Vision.