Paper FF-03

The Data Step; Your Key to Successful Processing In SAS®

Don Kros, Humana, Inc., Louisville, KY

ABSTRACT

The Data step is one of the basic yet more powerful of the building blocks in SAS programming language. Its power allows users to create the data sets that are used during a SAS program's analysis and reporting procedures. Understanding the basic structure, functioning, and components of the Data Step is fundamental to learning how to create your own SAS data sets. In this paper the focus will be on what a SAS Data Step is, why it is needed, how the Data Step works, and what information you can supply in your SAS Data Step to assist with getting the most out of a single data pass.

INTRODUCTION

"Why does it take so long for me to get my data extracted from SAS?" This seems to be the most common question asked when dealing with very large datasets. The issue is dramatically amplified when there are 450 SAS Users competing for extracts on tables with over 2 Billion rows and Two Hundred columns. Regardless of the type of hardware a company invests in to solve the problem, nothing will take the place of properly written code. Of course, properly written can mean something different to every person, for the purposes of this paper, properly written will focus on the Data Step. Why focus on the Data Step? The Data Step was designed to be the method used to extract data from SAS data sets. Proper use of the Data Step can provide a dramatic increase in user productivity.

UNDERSTANDING SAS PROCESSING WITH THE DATA STEP

Before we can explore any of the basic syntax and options in the Data Step, it is critical to gain an understanding of what is happening behind-the-scenes in SAS when a Data Step is executed.

To begin with, the Data Step consists of a group of SAS statements that begin with a Data statement. When you submit a Data step for execution, the complete syntax that makes up the Data Step is compiled, and the syntax is checked by SAS. If there is an error in the syntax, processing is stopped and an error is output in the SAS Log. If the syntax is correct, then the statements are executed. In its simplest form, the Data step is a loop with an automatic output and return action at the end of each cycle.

When the syntax review is complete and no errors are found, SAS creates an Input Buffer, a Program Data Vector, and Descriptor Information. Understanding each is important to understand the back end processing SAS completes when executing a Data Step:

The Input Buffer is a logical area in memory into which SAS reads each record of data from a raw data file when the program executes. It is acceptable to view the input buffer as temporary space SAS uses when reading a file that is not in SAS data set format.

When SAS reads from a SAS data set the data is written directly to the program data vector.

The Program Data Vector is a logical area of memory where SAS builds a data set, one observation at a time. When a program executes, SAS reads data values from the input buffer or creates them by executing SAS language statements. SAS assigns the values to the appropriate variables in the program data vector. From here, SAS writes the values to a SAS data set as a single observation.

The Descriptor Information is detail about each SAS data set, including data set attributes and variable attributes. SAS creates and maintains the descriptor information.

All executable statements in the Data Step are processed once for each observation. If the input file contains raw data, SAS reads a record into the Input Buffer. SAS then reads the values in the Input Buffer and assigns the values to the appropriate variables in the Program Data Vector. SAS also calculates values for variables created by program statements, and writes these values to the Program Data Vector. When the program reaches the end of the Data Step syntax for each observation, SAS loops back to the beginning of the Data Step for the next observation. Of interest is the fact that these actions occur by default. It is not necessary to submit a DO-LOOP or other syntax to create this process. This type of automatic processing makes using the SAS Data Step and other SAS syntax different from using most other programming languages.

THE DATA STEP ITSELF

Once you put some time into understanding how the Data Step is processed, you can summarize the flow into the following points;

1. SAS reads the syntax in the Data Step and applies the commands to the current observation in the Program Data Vector.
2. SAS writes the current observation from the Program Data Vector to the data set.
3. The program loops back to the top of the DATA step.
4. Variables in the program data vector are reset to missing values.
5. If there is another record to read, then the program executes again. SAS builds the second observation, and continues until there are no more records to read. The data set is then closed, and SAS goes on to the next bit of code to process.

An understanding of this simple process is the beginning of producing an accurate Data Step. When a Data Step is submitted it becomes clear how this record-by-record process, with an automatic loop simplify even complicated commands.

The following sections of this paper will focus on starting from a simple Data Step and build upon it. The examples are based on a fictitious SAS data set called ‘MEMBERSHIP’ which resides in a SAS Library called DATAPROD. This is an important fact because it is removing the need for the Input Buffer during the processing; this being because this data set is already a SAS data set. Another

assumption will be that before creating code, the user has a general understanding of the data and is able to execute a PROC CONTENTS statement to understand the size, compression and indexing used.

THE BASIC DATA STEP

If you were to run the Data Step in its simplest form that is without any data manipulation options like a 'WHERE' clause or an 'IF-THEN' Statement, you would create output data set that is identical in content to your input data set. Seldom is this the desired result of any data processing.

The Data Step contains three necessary statements:

- The DATA line identifies where you want to put data
- The SET line indicates where the data are coming from
- RUN indicates you wish to execute this process

This will begin the process of writing the code:

```
DATA Work.Membership;
SET Dataprod.Membership;
RUN;
```

As we discussed above, this process would read one line from the source data set, move it to the Program Data Vector, and output it to the specified data set. What will begin to develop in this process is that additional criteria can be added to the Data Step to begin manipulating the data to reach a desired output.

HOW ABOUT IF WE WANTED TO LIMIT OUR DATA TO A SPECIFIC TIME PERIOD?

The first process that can be examined is using the Data Step to subset information. Introducing a simple 'WHERE' clause to the statement is the easiest way to begin sub setting data. The 'WHERE' clause allows code to use variables within the data set to limit what information is retrieved. This allows the output data set to be more specific around the information we are interested in. Points to remember about the 'WHERE' clause are:

- SAS can build on the existing code we already have and add a 'WHERE' clause.
- A 'WHERE' clause tells SAS to only select specific data from the data set we are working with

This code would produce an exact copy of the MEMBERSHIP data but only for the CONTRACT_PERIOD 2008;

```
DATA Work.Membership;
SET Dataprod.Membership;
```

```
WHERE Contract_Period between '01JAN2008'd and '31DEC2008'd;
RUN;
```

A 'WHERE' clause does not need to be limited to a single variable in the data set. The clause can be used to create an output data set as specific as necessary by specifically identifying information within the' WHERE' clause based on the needs of the output data set.

HOW ABOUT MORE CRITERIA?

The 'WHERE' clause can be expanded in a number of different ways, for most purposes, using a 'AND' or an 'OR' in the statement is the most common. Adding additional criteria is important for two reasons. First, it allows the output data set to be specific in addressing the questions at hand. And second, the more specific the syntax is in stating what is necessary in the output, the faster the output will be generated. Of course it is possible to over-think what you decide to put in the 'WHERE' clause, making it less efficient when processing so it is important how the 'WHERE' clause is constructed.

To summarize:

- The more time and effort put into identifying the appropriate variables in the 'WHERE' statement, the faster you will get your data back.
- If the 'WHERE' clause includes multiple values from the dataset that are indexed, SAS processes the data much faster, because the indexing allows SAS find the data more quickly.

Here, the code has added to it three more criteria to our Data Step. This allows the code to be more specific on what it is SAS to provide;

```
DATA Work.Membership;
SET Dataprod.Membership;
WHERE Contract_Period between '01JAN2008'd and '31DEC2008'd
      AND Group ='01'
      AND Age >25
      AND Product ='AUTO';
RUN;
```

A BIT ON INDEXING

An index is a value that identifies and is used to locate a particular element, or elements within a data array or table. Indexing a data set, much like the index of a book, provides a fast way to locate information you are interested in without having to review every row of data. Not every element in a data set is, or should be indexed. Indexing data is expensive in both CPU Time and Disk Space. That being said, it is not realistic to think that a 'WHERE' statement will contain nothing but indexed values. But as a rule of thumb, it is best to include as many columns that are indexed as possible / necessary and still maintain the consistency of the data being created. For example, in this data set, assume the variable 'AGE' is not an indexed column, but the requirements are to

subset the population to members over 25, so the variable ‘AGE’ is included it in the statement. Indexing is a process that should be discussed between the Business and the Technical support for SAS. This way, the cost/benefit of indexing values can be explored.

SELECTING ONLY THE VARIABLES YOU WANT IN YOUR NEW TABLE

In the above code, the Data Step may have limited number of rows of data we are pulling back, but we have done nothing to limit the number of columns. Databases are constructed with the goal of holding information. Most commonly, data dealing with a specific topic are housed in a subset of tables within the database. The larger these tables become, the longer it takes to get meaningful information out of them. If asked to identify key performance issues within a database, one of the most common replies from a Database Administrator would be that the users of the database select more information from the database than is necessary. Processing data using a Data Step is no exception to this.

There are 2 ways to code in SAS specifying that you want only a subset of the information in your output. The first has already been accomplished by creating a query limiting the number of rows of data with a ‘WHERE’ clause. Now code will tell SAS to only populate the output data set with the columns necessary for the output data set. This can be done with either a ‘DROP=’ or a ‘KEEP=’ statement in the DATA line of our query. ‘DROP=’ will remove elements from the source dataset and ‘KEEP=’ will include elements from the source dataset. Decide which is most appropriate for the amount of data desired in the output data set. In this case, the code dictates using a ‘KEEP=’ statement because the output data set is only going to keep 11 columns of data. As a rule, apply the ‘KEEP=’ if the number of variables you want to keep is less than half the total number of variables in the dataset (n<N/2) otherwise use the ‘DROP=’. Limiting the amount of data you extract is an excellent way to speed up processing, as well as save disk space.

```
DATA Work.Membership (keep = Contract_Period Group Age Product
      Member_First Member_last Member_Adress Member_Phone Gender
      Deduct Auto_Class);
SET Dataprod.Membership;
WHERE Contract_Period between '01JAN2008'd and '31DEC2008'd
      AND Group ='01'
      AND Age >25
      AND Product ='AUTO';
RUN;
```

Limiting the number of columns selected for the output data set is not the only strategy available to save disk space, there are several others. One of the most common is compressing your data.

COMPRESSING YOUR DATASETS TO SAVE SPACE

Creating a well-constructed data set takes time. In an analytic environment, often the data sets being constructed are considered disposable. That is, they are created for a

specific purpose and when that purpose is no longer relevant, the data are discarded so disk space can be freed up for the next process. Knowing that the usefulness of a data set may be measured in hours or days rather than years, rules for the creation of a proper data set are often ignored. This can usually lead to the creation of large and unruly data sets. Storing only the necessary variables is an excellent method for limiting the size of a data set, however applying a compression statement is as, if not more, effective in saving space. Adding the syntax 'COMPRESS=YES' to the data line allows SAS to make a determination if the dataset would benefit from being compressed. In some cases, it would not, and the compress function will not be applied. When dealing with large datasets SAS can save as much as 60% of the otherwise used space by including the 'COMPRESS=YES' syntax.

Compression is not without cost. Compressing data does require CPU and processing time. However in most cases the expense is justified when compared to the savings. With limited worry on the overhead of processing the 'COMPRESS=YES' syntax, compressing your datasets would be considered good practice to implement.

In this code set, the 'COMPRESS=YES' statement is added in front of the 'KEEP=' syntax:

```
DATA Work.Membership (compress=yes
      keep = Contract_Period Group Age Product
      Member_First Member_last Member_Adress Member_Phone Gender
      Deduct Auto_Class);
SET Dataprod.Membership;
WHERE Contract_Period between '01JAN2008'd and '31DEC2008'd
      AND Group ='01'
      AND Age >25
      AND Product ='AUTO';
RUN;
```

The compression syntax can be added in an 'OPTIONS' line as well. Including the syntax in a Data Step will apply the compression to the dataset you are creating, where using an OPTIONS statement will compress all data sets created in that SAS Session.

RENAMING VARIABLES IN YOUR DATASETS

Data are not always set up for reporting needs or analytic consumers. Often time data elements are given cryptic names that can make them confusing to work with. As with SQL, there is simple syntax in the Data Step that you follow to accomplish this; (RENAME=(OLD=NEW)). In the example SAS is renaming the variable 'Deduct' to 'Member_Deductible' by including the syntax (Rename = (Deduct = Member_Deductible)). Within the code, SAS can rename as many variables as necessary, however when using the 'DROP=' statement, it is important to remember to not drop and rename the same variable in the same statement. In the example, note how the new variable 'Member_Deductible' was added to the 'KEEP=' statement on the DATA line. It may be redundant but is consider best practice to always include any variable you expect to be contained in the output data set in the 'KEEP=' on the DATA statement.

The Data Step is taking shape with these additional codes and now looks like:

```
DATA Work.Membership (compress=yes
      rename=(Deduct=Member_Deductible)
      keep = Contract_Period Group Age Product
      Member_First Member_last Member_Adress Member_Phone Gender
      Deduct Auto_Class Member_Deductible);
SET Dataprod.Membership;
WHERE Contract_Period between '01JAN2008'd and '31DEC2008'd
      AND Group ='01'
      AND Age >25
      AND Product ='AUTO';
RUN;
```

SELECTING ONLY THE VARIABLES NECESSARY FOR PROCESSING

In a previous step, the syntax included a 'KEEP=' statement on the DATA line. This syntax limited the number of variables we included in the output data set. The Data Step offers another method for using the 'KEEP=' or DROP=' statement. Within the Data Step, SAS can limit the number of columns read off of the source data set and placed into the Program Data Vector and subsequently what is available for the 'KEEP=' or 'DROP=' used to limit the output data set. This is done by putting a 'KEEP=' or a 'DROP=' in the SET statement. The same formula of (n<N/2) can apply as well.

Because this syntax is limiting what SAS processes, it is important to include all output variables desired in the output data set that are on the source data set, all variables from the 'WHERE' clause, and any variables necessary for further processing. If the 'KEEP=' data set option is associated with a source data set, only those variables that are listed after the 'KEEP=' data set option are available for processing. If the 'KEEP=' data set option is associated with an output data set, only the variables listed after the option are written to the output data set, but all variables are available for processing.

Note that in this example, all the elements in the SET 'KEEP=' statement are *NOT* in the DATA 'KEEP=' statement. The variables used as part of the WHERE statement have been removed from the 'KEEP=' in the DATA line and thus will not become part of the output data set. The key concept is that by limiting the number of columns SAS processes from the source data set there is an expected increase in the speed with which SAS will generate a resulting data set.

```
DATA Work.Membership (compress=yes
      rename=(Deduct=Member_Deductible)
      keep = Member_First Member_last Member_Adress
      Member_Phone Gender Auto_Class Member_Deductible);
SET Dataprod.Membership (keep = Contract_Period Group Age Product
      Member_First Member_last Member_Adress Member_Phone Gender
      Deduct Auto_Class Deduct);
WHERE Contract_Period between '01JAN2008'd and '31DEC2008'd
      AND Group ='01'
      AND Age >25
```

```
      AND Product ='AUTO';
RUN;
```

CREATING NEW VARIABLES AND FORMATS WITH IF-THEN STATEMENTS

Up to this point, the focus has been on working with existing variables. Often times when working with data, it is necessary to create additional variable that are useful in reporting or analytic processes. When it becomes necessary to create a new variable it is possible to do so in the same Data Step.

In this example, the code will create a variable off of the existing AGE category. Because of the existing syntax in the WHERE clause each person selected has an age greater than twenty-five but it might make things easier if there were a few age buckets that could use later in summarizing the data. To accomplish this, the first step is to create a variable called 'BUCKET' and tell SAS it is 10 characters in length. Then put conditions around what is going to populate the new field.

This is a rather simple example, but the concept holds true regardless of how complicated the statement needs to be. Within a single Data Step, it is possible create as many variables and applied criteria as necessary for analysis. Any variable available from the source data set is able to be used as part of the criteria to create a new variable. Note how the new variable 'BUCKET' was added to the 'KEEP=' statement on the DATA line. It may be redundant but is considered to be a best practice to always include any variable expected to be contained in the output data set in the 'KEEP=' on the DATA statement.

```
DATA Work.Membership (compress=yes
      rename=(Deduct=Member_Deductible)
      keep = Member_First Member_last Member_Adress
      Member_Phone Gender Auto_Class Member_Deductible
      Cost  Bucket);
SET Dataprod.Membership (keep = Contract_Period Group Age Product
      Member_First Member_last Member_Adress Member_Phone Gender
      Deduct Auto_Class Deduct Cost);
WHERE Contract_Period between '01JAN2008'd and '31DEC2008'd
      AND Group ='01'
      AND Age >25
      AND Product ='AUTO';
FORMAT Bucket $10.;
IF age < =40 THEN Bucket ='LEVEL1'; ELSE
IF 40 > age < 55 THEN Bucket=' LEVEL2'; ELSE
Bucket =' LEVEL3';
RUN;
```

CREATING AN EQUATION WHILE IN THE DATA STEP

Much the same way SAS created the new variable 'BUCKET' SAS can create a variable, assign it a numeric format, and make it contain the result of an expression. In this example the syntax is set up to create a variable called 'COST', format it as a number and then sum the existing variables Deduct and Paid as the expression. Note

that the formats of the ‘BUCKET’ and the ‘COST’ are different. ‘BUCKET’ is text where ‘COST’ is numeric. The format of a new variable needs to be assigned when the variable is created. If a format is not set, the variable will take on the format and size of the first data element assigned to it.

```
DATA Work.Membership (compress=yes
      rename=(Deduct=Member_Deductible)
      keep = Member_First Member_last Member_Adress
      Member_Phone Gender Auto_Class Member_Deductible
      Paid Bucket Cost);
SET Dataprod.Membership (keep = Contract_Period Group Age Product
      Member_First Member_last Member_Adress Member_Phone Gender
      Deduct Auto_Class Deduct);
WHERE Contract_Period between '01JAN2008'd and '31DEC2008'd
      AND Group ='01'
      AND Age >25
      AND Product ='AUTO';
FORMAT Bucket $10.;
FORMAT Cost 8.2;
IF age < =40 THEN Bucket ='LEVEL1'; ELSE
IF 40 > age < 55 THEN Bucket=' LEVEL2'; ELSE
Bucket =' LEVEL3';
Cost = (Deduct + Paid);
RUN;
```

CREATING NEW DATE FORMATS WHILE IN THE DATA STEP

Working with data elements related to dates can prove to be a daunting task. Among the most common requests deal with finding the time span between two dates and rolling up sums or counts from time spans ranging from Seconds to Years. To make working with dates easier, syntax can be included to create multiple formats within the output data set allowing for faster calculations when working with dates in further analysis.

In this code, the objective is to create two additional variables off of the ‘CONTRACT_PERIOD’ variable, specifically to create a ‘MONTH’ and a ‘YEAR’ variable allowing for ease in summing up data easier in later analysis. The first step is to create the new variables and give them the correct formats. After that, syntax tells SAS that the existing date should adopt the new format. Finally, as previously mentioned, put the new variables in our ‘KEEP=’ statement in the DATA line.

```
DATA Work.Membership (compress=yes
      rename=(Deduct=Member_Deductible)
      keep = Member_First Member_last Member_Adress
      Member_Phone Gender Deduct Auto_Class Member_Deductible
      Paid Bucket Cost Month Year);
SET Dataprod.Membership (keep = Contract_Period Group Age Product
      Member_First Member_last Member_Adress Member_Phone Gender
      Deduct Auto_Class Deduct);
WHERE Contract_Period between '01JAN2008'd and '31DEC2009'd
      AND Group ='01'
      AND Age >25
```

```
      AND Product ='AUTO';
FORMAT Bucket $10.;
FORMAT Cost 8.2;
FORMAT Month monyy7.;
FORMAT Year year4.;
IF age < =40 THEN Bucket ='LEVEL1'; ELSE
IF 40 > age < 55 THEN Bucket=' LEVEL2'; ELSE
Bucket =' LEVEL3';
Cost = (Deduct + Paid);
Month = Contract_Period;
Year=Contract_Period;
RUN;
```

This example only begins to describe the functions available when dealing with Dates and Times. Depending on the needs, this may be enough to get started with working with dates, or a realization that for any other data, having a much more in-dept working knowledge of Dates and Times in SAS is necessary. Regardless, understanding the basics is a necessity.

CREATING MULTIPLE OUTPUT TABLES FROM A SINGLE DATA STEP

To this point, the focus has been on working with Syntax that will create a subset of data from our input data set. Sometimes it may be necessary to subset resulting data sets before you begin a specific type of analysis. In other instances, it may be necessary to run code multiple times making changes to the code to meet a different need each time. Within the Data Step, syntax can be added that allow the code to create multiple outputs from a single instance of code.

To demonstrate this, the variable ‘PRODUCT_CAT’, which is a subset from within in the ‘PRODUCT’ variable, will provide the necessary information. Metadata would state that there are several codes, ‘SPORT’, ‘SUV’, ’RV’ etc. which are ‘PRODUCT_CAT’ codes that further describe the variable ’AUTO’. For demonstration purposes, syntax will group them into 3 buckets and output them into not just the single WORK table, but 3 different WORK tables depending on the ‘PRODUCT_CAT’ variable on each line of data.

An important note to understand is SAS must define the output tables in the original DATA line syntax. It is not necessary to keep the same variables on the ‘KEEP=’ statement as in this example, however any variable needed in the outputs do need to be in the ‘KEEP=’ in your SET statement. Sub-setting criteria does not have to be limited to a single variable like ‘PRODUCT_CAT’ or ‘AGE’, SAS can create tables and additional variables off any combination of variables contained within the original data set.

```
DATA Work.Work.Cars (compress=yes
      rename=(Deduct=Member_Deductible)
      keep = Member_First Member_last Member_Adress
      Member_Phone Gender Deduct Auto_Class Member_Deductible
      Paid Bucket Cost Month Year)
            Work.Work.Trucks (compress=yes
      rename=(Deduct=Member_Deductible)
```

```
      keep = Member_First Member_last Member_Adress
      Member_Phone Gender Deduct Auto_Class Member_Deductible
      Paid Bucket Cost Month Year)
            Work.Work.Large_Vehicle (compress=yes
      rename=(Deduct=Member_Deductible)
      keep = Member_First Member_last Member_Adress
      Member_Phone Gender Deduct Auto_Class Member_Deductible
      Paid Bucket Cost Month Year);
SET Dataprod.Membership (keep = Contract_Period Group Age Product
      Product_Cat Member_First Member_last Member_Adress Member_Phone Gender
      Deduct Auto_Class Deduct);
WHERE Contract_Period between '01JAN2008'd and '31DEC2009'd
      AND Group ='01'
      AND Age >25
      AND Product ='AUTO';
FORMAT Bucket $10.;
FORMAT Cost 8.2;
FORMAT Month monyy7.;
FORMAT Year year4.;
IF age < =40 THEN Bucket ='LEVEL1'; ELSE
IF 40 > age < 55 THEN Bucket=' LEVEL2'; ELSE
Bucket =' LEVEL3';
Cost = (Deduct + Paid);
Month = Contract_Period;
Year=Contract_Period;
IF Product_Cat in ('SPORT','UTILITY','SEDAN') then output Work.Cars;
IF Product_Cat in ('SUV','TRUCK') then output Work.Trucks;
IF Product_Cat in ('RV','SEMI') then output Work.Large_Vehicle;
RUN;
```

CONCLUSION

The SAS Data Step is the foundation upon which SAS programming is built. The Data Statement is responsible for defining the variables used in your program, reading each data case and assigning values to the variables, creating new variables, transforming values, formatting and labeling variables, and assignment of missing values. Without an understanding of how the DATA Statement works it is nearly impossible to use SAS successfully.

In summary, key points to remember about the SAS Data Step are:

- The Data Step is the best way to grab data out of a SAS Data Set because it is designed specifically for that purpose.
- When creating a Data Step, you have quite a bit of freedom in what you do with the Data Step to create what you need as you are processing.
- The Data Step reads through the data only one time and applies all the criteria you write to each line of data you are interested in.
- Data Steps are very easy to troubleshoot with the LOG files

- Finally, the more complicated a Data Step is, the harder SAS is going to need to work. Harder work in SAS results in more processors and more threads. To the user, this means faster results.

This paper has covered a few key aspects of the Data Step, but be aware that there is much more the Data Step can do. Please visit www.support.sas.com or your favorite Internet Search Engine to continue learning about the SAS Data Step

ACKNOWLEDGMENTS
I would like to thank Lisa Kros for her assistance and encouragement while writing this paper.

CONTACT INFORMATION

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

Name	Don Kros
Enterprise	Humana Inc
Address	500 W. Main Street
City, State, ZIP	Louisville, KY, 40202
Work Phone:	502.476.1134
E-mail:	dkros@humana.com -or- Don.Kros@insightbb.com
Web:	www.humana.com Or find me at www.linkedin.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.
Other brand and product names are trademarks of their respective companies