Paper 061-30

Slicing and Dicing the SAS® Data Set

William C. Murphy
Howard M. Proskin & Associates, Inc., Rochester, NY

ABSTRACT

Everyone knows how to get all of the information from one SAS data set into another: just 'SET' the source data set to the output data set in a DATA step. Often all the observations in the source data set are not needed or desired. Most programmers would then use the FIRSTOBS and OBS options to limit the range of observations that they would input from the source data set. However, the SET statement has the ability to acquire data from the source data set in a non-sequential fashion. This random access of the source data set is accomplished with the POINT option on the SET statement. With this option, you can 'point' to and read in any observation in any order from the source and include the observation in the output data set. Using this feature, small data sets can be created from huge data sets for program testing and debugging. In merging data, the POINT option can be used to combine data sets in ways that exceeds the ability of the MERGE statement. When doing repetitive tasks with a macro, use of the POINT option can limit the number of macro variables. In general, the POINT option on the SET statement is a useful programming tool for saving time and resources.

INTRODUCTION

Where the SAS System is employed, you will find daily use of the DATA step. Programmers are constantly reading in one SAS data set and outputting another within the DATA step. Except for subsetting the input data set with WHERE statements, most programmers read all the observations in the input DATA step. When they limit the observations, the programmers may just select some starting and/or ending observation, but still read the lines between these observations sequentially. The SET statement, however, has the ability to select any observation for input. In the following discussion, we will show how this selective inputting can be used to create sample data sets, combine data sets, and streamline macro programs.

THE DATA STEP

To read one data set into another, we can simply write

```
data Out;
       set Source;
       run;
```

where the SAS system will read observations from the data set Source and output them into the data set Out, starting with the first observation in Source and proceeding sequentially until the last observation is read .

If you want to ignore the first few observations in the Source data set, you can employ the FIRSTOBS= option;

```
data Out;
       set Source (firstobs=3);
       run;
```

Now the DATA step will start reading Source with the observation number 3. Similarly, we can choose the last observation to be read by employing the OBS= option:

```
data Out;
       set Source(firstobs=3 obs=10);
       run;
```

where the DATA step will now read the Source data set starting with observation number 3 and ending with observation 10. You can use one or both of the options with your input data set as long as the value of FIRSTOBS= is less than or equal to value of OBS= and as long as the selected observations exist in the data set.

These options do give us the ability to select any range of observations. However, they still limit us in reading the data set sequentially. If we wanted to read, for example, observations 9 through 15 and then read observations 3 through 5, these simple options will not work. The SET statement however does have the ability to select the input in any order.

THE POINT= OPTION

If we wanted to read only observation 17 in the Source data set, we could set the values of both the FIRSTOBS= and OBS= options to 17. However, the SET statement is provided with a POINT= option that allows us to point to and select a given observation:

```
data Out;
       Slice=17;
       set Source point=Slice;
       output;
       stop;
       run;
```

This code will gives us only observation 17 from Source in the Out data set. You may wonder why we used what seems to be extra code in addition to the POINT= option. We needed this code to account for certain peculiarities of this option: the POINT= option must be set equal to a temporary variable , not a constant number, and, since the data is being randomly accessed (*i.e*. no end-of-file indicator is encountered), you must tell the DATA step when to 'OUTPUT' and when to 'STOP'.

In order to slice more observations from the input data set, the POINT= option can be combined with a DO loop:

```
data Out;
       do Slice=17, 1, 23, 17, 5, 6;
              set Source point=Slice;
              output;
              end;
       stop;
       run;
```

This code shows that the POINT= option can be used to access any observation, any number of times, in any order from the input data set. This ability to randomly access the input data set can be applied in a number of ways to ease programming chores.

SAMPLING DATA SETS

In nearly all of the work that I have been involved with, I have used the SAS system on a PC. In most cases, I deal with data sets with a few thousand or less observations and maybe two dozen variables. With such data sets, programming and debugging on the PC is a breeze. But occasionally I have been confronted with medical records for the community which involved tens of millions of observations and hundreds of variables. For such data sets, you want to run a program only once since it may take hours. To develop the programs, therefore, you want to work on only a small subset of the main data set. You could just take the first thousand records for instance, but there may be some bias in the way the data set was formed. Instead of this technique, we can use the SET statement with the POINT= option to sample the data:

```
data Sample;
       do i=1 to 1000;
              Slice=int(nObs*ranuni(123456789));
              set BigDataSet point=Slice nobs=nObs;
              output;
              end;
       stop;
       run;
```

where we use the SAS function RANUNI to choose randomly a thousand observations from BigDataSet. These observations will be uniformly distributed over the total number of observations in BigDataSet, obtained with the option NOBS=. (The NOBS= option is done before execution so that nObs is available before the SET statement executes.) The resulting data set Sample is much easier to use for program development and debugging. Depending on your need, you can sample as many observations as you like using a variety of random number generating functions that the SAS system provides.

COMBINING DATA SETS

When you have two data sets with some common identification variable, combining them into one data set with the MERGE statement is a simple chore. For example, if we combined data from a physical exam with data on the person drug history, we would have

```
data Medical;
       merge Physical Drugs;
       by Patient;
       run;
```

As long as the patient has one observation in Physical or Drugs and any number in the other data set, this procedure would work. If the patient had several physicals with an observation in Physical for each and an associated drug list with several entries, this code would produce a warning in the log about both data sets having multiple observations for the same Patient. The resulting data set Medical would in general not be what you wanted. Advocates of PROC SQL would join these data sets with no problem. However, even using DATA step coding, we can merge these data sets using the SET statement with the POINT= option:

```
data Medical;
       set Physical (rename=(Patient=tmpPatient));
       do i=1 to nObs;
              set Drugs nobs=nObs point=i;
              if Patient=tmpPatient then output;
              end;
       run;
```

This code is merging the data in a very simple way and avoids the problems associated with the MERGE statement. The first SET statement causes the DATA step to input, one by one in sequential fashion from the data set Physical. For each observation read in from Physical, the DO loop is executed. This loop uses the POINT= option on another SET statement to go through each observation in Drugs and outputs the data only if the Patient identification corresponds to the Patient identification of the Physical observation. Thus each observation of Physical can have several observations from Drugs attached to it. Since several observations in Physical can be for the same Patient, the same observations from Drugs will be attached to these common Patient observations. In other word, we have a many-to-many observations merge. Two further points should be noted: The variable Patient had to be renamed so that the same variable from the second SET statement did not over write it and, as in our data set sampling example, the NOBS= option is done before execution so that the DO loop can precede the SET statement.

MACRO SAVINGS

If you were given a SAS data set that contained a list of variables and they involved identical analyses, you might use the macro facility of the SAS system to process them. You might for example read the variables into macro variables:

```
data _null_;
       set VarList;
       call symput('Vrbl'||_n_,VarForAnalysis);
       run;
```

where we have used SYMPUT and the automatic variable _n_ to create the macro variables &Vrbl1, &Vrbl2, &Vrbl3, *etc*. from the list of variables contained in the data set VarList. You would then write a macro %DO loop to process them:

```
%do i=1 %to 10;
       %put ***** Processing &&Vrbl&i *****;
```

```
                ...

        ( Lots of PROCs and DATA steps )

                ...
        %end;
```

where we have assumed there were 10 variables for processing. If we had 100 or 1,000 or 10,000, we just might keep making more macro variables and expanding the %DO loop. We could even add custom titles and footnotes to out VarList data set and subsequently, create macro variables for them and evoke them in the %DO loop. However, you might find that you are quickly eating up your systems memory resources.

This problem could be avoided if we employed the POINT= option on the SET statement and rewrote the code:

```
%do i=1 %to 10;

        data _null_;
              Choice=&i;
              set VarList point=Choice;
              call symput('Vrbl',VarForAnalysis);
              run;

        %put ***** Processing &Vrbl *****;
                ...

        ( Lots of PROCs and DATA steps )

                ...
        %end;
```

where we now have included the _NULL_ DATA step inside our macro %DO loop. Each time the DATA step is executed only one observation is read, controlled by the macro %DO loop index &i. The macro variable &Vrbl no longer needs the &i suffix since only one variable is being processed at a time. Thus we do not need to create macro variables for all observations in Varlist, regardless of how many observations there are. Therefore no memory is wasted on excess macro variables.

CONCLUSION

Accessing data from a SAS data set in a DATA step can be done two ways. The first and more common way is sequentially inputting all of the data. The second way is to access the data in a random way according to dictates of the project. This second way is accomplished with a SET statement containing a POINT= option. This technique is far more powerful than the sequential access method. It can be used to sample large data sets, do many-to-many merges, and save on macro variables. In general, the POINT= method can be used in many cases to save both time and memory by limiting processing to only desired observations.

REFERENCES

Aster, Rick, and Seidman, Rhena (1991). *Professional SAS® Programming Secrets*. Windcrest Books, Blue Ridge Summit, PA, p. 165, 415.

Aster, Rick (2002). *Professional SAS® Programming Shortcuts*. Breakfast Communications Corporation, Paoli, PA, p. 351.

Riba, David S. (1998). "The SET Statement and Beyond: Uses and Abuses of the SET Statement". *Proceedings of the Twenty-Third Annual SAS® Users Group International Conference*, SAS Institute Inc., Cary, NC, p. 293.

SAS Institute Inc. (2002). "Set Statement". *SAS OnLineDoc® 9*. SAS Institute Inc., Cary, NC. (url: http://v9doc.sas.com/sasdoc).

CONTACT INFORMATION

Your comments and questions are values and encouraged. Contact the author at

William C. Murphy
Howard M. Proskin & Associates, Inc.
300 Red Creek Dr., Suite 330
Rochester, NY 14623
Phone 585-359-2420
FAX 585-359-0465
Email wmurphy@hmproskin.com or wcmurphy@usa.net
Web www.hmproskin.com