This paper and presentation are designed to provide some basic fundamentals on SAS datasets and how they can be manipulated using the SET and MERGE statements. The intention is to give novice users a guide which can be used as a quick-reference tool.

As a first step, it is important to realize that SET and MERGE are only valid with existing SAS datasets. It is necessary to first describe SAS datasets and how they are created before any discussion on their manipulation can be accomplished.

Every SAS program, no matter how difficult, is divided into two parts, as shown in the following diagram:

As the diagram describes, SAS datasets are created in the Data step, where all data manipulations occur as well. The Proc step then processes all data using the many different PROCs available in the base SAS system. How to create SAS datasets can be accomplished in two different ways using the INPUT statement. If a non-SAS dataset exists (such as text file, for example) then the INPUT statement can be used along with the INFILE statement, which will open the file and read the data, and upon execution create a SAS dataset. If the data are created in-stream then the INPUT statement must be used with the CARDS statement, followed by the data to be used. This process will read the data and create the resulting SAS dataset. This second method is the one to be used as examples throughout this paper. For more information on INPUT and how to input data, refer to the SAS Users' Guide.

A quick synopsis of SAS Datasets follows:

SAS Datasets:
- Particular type of file
- Created only by a SAS program
- Collection of data values arranged in rectangular form
- Number of datasets is limited only by space requirement
- Lots of hidden characters
- Contains information on the data which is revealed upon execution of a PROC CONTENTS

Once a SAS dataset has been created, various SAS statements can be used to manipulate the dataset; the concentration here is on the SET and MERGE statements.
The SET statement:

- reads observations from one or more SAS datasets
- used to copy, concatenate, interleave, or subset observations from existing SAS dataset(s) into a new SAS dataset
- all variables are included unless DROP= or KEEP= options are used
- up to 50 datasets
- can use permanent SAS datasets or temporary datasets or a combination of the two
- SET; => if dataset name is omitted, the system will create a special name _LAST_ which will refer to the SAS dataset most recently created.

To copy a SAS dataset, consider the following example:

```sas
data one; input @1 name $10. @11 account 4. @16 site $6. @23 code $5. @29 age 2.;
cards;
Richard 007 main V70.0 28
Theresa 1001 clinic 844- 32
Beth 3500 clinic 487- 31
Paul 1500 main 461 29
;
data two; set one;
proc print;
title 'Copy a SAS Dataset'; run;
```

The above program simply creates a SAS dataset in Data One and copies it into Data Two. The output is displayed with all variables and observations in the same order as they were created.

To concatenate two SAS datasets, consider the following example:

```sas
data one; input @1 name $10. @11 account 4. @16 site $6. @23 code $5. @29 age 2.;
cards;
Scott 800 main 477- 27
Paul 1500 main 461 29
;
data two; input @1 name $10. @11 account 4. @16 site $6. @23 code $5. @29 age 2.;
cards;
Mike 1190 clinic 460- 29
Lori 1191 clinic 460- 29
Mark 4320 V70.0
;
data three; set one two; proc print;
title 'Concatenate two SAS Datasets'; run;
```

Note that the SET statement in Data Three simply names the two SAS datasets being concatenated. The result is one dataset composed of all observations and variables of Data One, followed by all observations and variables of Data Two. No "mixing" of data of any kind occurs. Also note the representation of missing data in the output:

```
OBS  NAME  ACCOUNT  SITE  CODE  AGE
1  Scott  800  main  477-  27
2  Paul  1500  main  461  29
3  Mike  1190  clinic  460-  29
4  Lori  1191  clinic  460-  29
5  Mark  4320  V70.0
```

To interleave SAS datasets means to combine two or more SAS datasets in some sorted order. This is done by using the BY statement in the dataset where the interleave takes place. The BY requires a variable, which is the variable that is
used for the sort. One stipulation with interleaving is that each dataset being used for the interleave must be sorted by the same variable that is specified with the BY statement; this is easily accomplished by using the PROC SORT in each dataset if necessary. Otherwise the interleave will not work. The following example illustrates this:

```sas
data one; input @1 name $10. @11 account 4. @16 site $6. @23 code $5. @29 age 2.; cards;
   Richard 007 main V70.0 28
   Theresa 1001 clinic 844- 32
;
data two; input @1 name $10. @11 account 4. @16 site $6. @23 code $5. @29 age 2.;
cards;
   Steve 2303 clinic V70.0 28
   Beth 3500 clinic 487- 31
   Paul 1500 main 461 29
;
proc sort; by name;
data three; set one two; by name;
proc print; title 'Example of Interleave'; run;
```

The idea is to combine Data One and Data Two in alphabetical order (sorted by name). So, in Data Three, the SET is used with the by name; statement ('name' being the variable to be sorted). When looking at Data One, note that the names are already sorted; no PROC SORT is required here. However, the names are not sorted in Data Two, so a PROC SORT is needed in this dataset. The result in Data Three:

```
Example of Interleave

<table>
<thead>
<tr>
<th>OBS</th>
<th>NAME</th>
<th>ACCOUNT</th>
<th>SITE</th>
<th>CODE</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Beth</td>
<td>3500</td>
<td>clinic</td>
<td>487-</td>
<td>31</td>
</tr>
<tr>
<td>2</td>
<td>Paul</td>
<td>1500</td>
<td>main</td>
<td>461</td>
<td>29</td>
</tr>
<tr>
<td>3</td>
<td>Richard</td>
<td>7</td>
<td>main</td>
<td>V70.0</td>
<td>28</td>
</tr>
<tr>
<td>4</td>
<td>Steve</td>
<td>2303</td>
<td>clinic</td>
<td>V70.0</td>
<td>28</td>
</tr>
<tr>
<td>5</td>
<td>Theresa</td>
<td>1001</td>
<td>clinic</td>
<td>844-</td>
<td>32</td>
</tr>
</tbody>
</table>
```

**Subsetting Variables and Observations**

This can be done by using some of the options that are available with the SET statement. The number of the variables to be processed can be controlled with the KEEP and DROP; the number of observations to be processed can be controlled by using the FIRSTOBS= and OBS= options.

There are three methods for using the KEEP and DROP, but only one is truly an option of the SET statement. One other is an option in the data step and the third is simply a statement in the data step. There are differences between the three; however, the focus here is on how to the SET option.

Assuming that a Data One exists with at least variables A, B, C, D:

```sas
Data Two;
Set One (KEEP= A B C D);
```

As a SET option, the subset is created before the data are brought into the data step. Hence, in the above example, only variables A, B, C, and D are available for processing in Data Two.

KEEP and DROP, as one might expect, accomplish opposite tasks. Use the KEEP whenever a variable is to be processed; use the DROP whenever a variable is not to be processed. Remember that this is only one way of using the KEEP and DROP; there are three methods, and a suggested way to see how the three samples differ is to write a simple program and run it, using the three methods. Check the SAS Log as well as the output.

The OBS= option allows for a given number of observations to be processed, beginning with the first observation. The FIRSTOBS= option allows
for processing to begin at a particular observation other than the first (thus the difference between the two options is that OBS= is a total amount of observations to be processed beginning with the first observation, while the FIRSTOBS= option designates a particular observation from which to begin processing).

It is possible to use multiple options with a SET statement, and combining the ones described above can be very powerful in reducing the time required to run programs because the size of datasets processed is also reduced. Consider the following example:

```sas
data one; input @1 name $10. @11 age 2. @14 height $5. @20 weight 3. @24 hair $6. @31 eyes $5.;
cards;
Richard 28 6'0" 138 brown brown
Beth 31 5'7" 135 grey blue
Steve 28 5'9" 155 brown blue
Cindy 27 5'5" 140 brown brown
Paul 29 5'8" 160 blonde blue
Scott 27 5'8" 145 blonde blue
Mike 29 5'6" 150 blonde blue
Lori 29 5'3" 130 blonde blue
Mark 28 6'3" 180 blonde brown;
```

The above would reduce both the number of observations to be processed as well as the variables, as shown by the output:

```
Sub-SET the SAS Dataset

<table>
<thead>
<tr>
<th>OBS</th>
<th>NAME</th>
<th>HEIGHT</th>
<th>WEIGHT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Beth</td>
<td>5'7&quot;</td>
<td>135</td>
</tr>
<tr>
<td>2</td>
<td>Steve</td>
<td>5'10&quot;</td>
<td>155</td>
</tr>
<tr>
<td>3</td>
<td>Cindy</td>
<td>5'9&quot;</td>
<td>140</td>
</tr>
</tbody>
</table>
```

Remember that OBS= begins at the first observation. In the above case, OBS=5 designates the first 5 observations. But since FIRSTOBS=3 is declared, only observations 3, 4, and 5 are included for processing.

There are several other options available with the SET statement that are not covered here. Refer to the SAS Users' Guide for more information.

MERGE is an operation performed on two existing SAS datasets to join two observations into one. A synopsis of MERGE follows:

```
MERGE
- joins observations from two or more SAS datasets into a single observation in a new SAS dataset
- can merge temporary and permanent SAS datasets
```

Suppose it was not necessary to process all these observations and all these variables. Suppose all that is needed is to process three of the variables: name, height, and weight. Furthermore, suppose it was only required to begin processing at the third observation and that three observations are to be processed. Take a look at the corresponding SET options:

```sas
data two; set one(firstobs=3 obs=5 keep= name height weight);
proc print; title 'Sub-SET the SAS Dataset'; run;
```
2 types of Merging

one-to-one (no BY) (observation by observation)

BY statement (match-merge)

- only one BY statement
- based on presence of variable
- all datasets must have the BY variable

if # of observations or variables are not same then missing values are used as needed

When merging one-to-one it is important to remember that the first observation from one dataset is merged with the first from the second dataset no matter what information is contained. That means if there are variables in one observation that do not exist in the other, then missing values are used to represent these different variables. Similarly, if one dataset has more observations than the other, then the extra observations are simply added to the dataset.

One of the possible pitfalls when doing one-to-one merging is shown in the next example:

```
data one; input $1 name $10. $11 age 2. $14 height $6.;
cards;
Anthony 28 6'0"
Yvette 28 5'8"
SCOTT 28 5'6"

data two; input 2 age 2. 3 hair $6. 4 eyes $5. 11 name $10.;
cards;
25 brown brown Yvette
32 brown brown Anthony
blonde brown SCOTT

data three; merge one two;
proc print; title 'MERGE obs by obs- what a mess'; run;
```

The data shows the two SAS datasets to be created, with the merge to take place in Data Three. The output:

```
MERGE obs by obs- what a mess
OBS NAME AGE HEIGHT HAIR EYES
 1 Yvette 25 bROWN brown brown
 2 Anthony 32 5'8" brown brown brown
 3 Scott . 5'6" blonde blue
```

Note that while the variables 'name' and 'age' appear in both datasets, only the values in Data Two appear in the merged dataset. Any variable appearing in Data One has its value replaced by the value of the corresponding variable in Data Two, on an observation-by-observation basis. This is a result of the one-to-one MERGE, and can present problems regarding the integrity of data if one is not careful (for example, Yvette is a friend of mine and she is not 6'0" although the output dataset says she is). Another result of the MERGE (one-to-one or with a BY) would be that if a variable is present in Data One but not in Data Two (or vice versa), then those values remain intact. This is also evident in the above example.

It is possible to merge using a BY statement, in order to merge in sorted order. The stipulations
using the BY with the MERGE are the same as with the SET (if the datasets are not pre-sorted, use a PROC SORT to accomplish the task).

```sas
data one; input @1 name $10. @11 age 2. @14 height $6.;
cards;
Anthony 28 6'0"
Yvette 28 5'8"
Scott 28 5'6"
30 5'7"
;
proc sort; by name;
```

```sas
data two; input @1 age 2. @4 hair $6. @11 eyes $5. @17 name $10.;
cards;
28 brown brown Yvette
32 blonde blue Anthony
28 blonde blue Scott
27 brown brown Cindy
;
proc sort; by name;
```

```sas
data three; merge one two; by name;
```

```sas
proc print; title 'MERGE by name';
run;
```

The corresponding output:

```
MERGE by name

OBS  NAME  AGE  HEIGHT  HAIR  EYES
    1     .    .      .      .      .
    2  Anthony 32  6'0"  blonde  blue
    3     Cindy 27    5'7"  brown  brown
    4  SCOTT  28  5'6"  blonde  blue
    5  Scott  28    5'8"  blonde  blue
    6  Yvette 28    5'7"  brown  brown
```

Several things to note here. First, the names are indeed in alphabetical order. Because the data in Data One and Data Two were not already sorted, a PROC SORT had to be included in those datasets. The other thing to keep in mind is that in PC SAS, upper case characters have a lower value than lower case characters. As the output indicates, 'SCOTT' and 'Scott' are not the same. This may differ depending on the operating system; users should try this out on their system to see how the evaluation occurs. Also, in Data One there is a name with a missing value; this is the first observation in the output. So in the above example, even though each dataset had four observations, a total of six observations were created in Data Three.

There are several other features related to SET and MERGE that are not covered here; however it is my hope that this helps serve as a basis to novice users of the SAS system in the creation and manipulation of SAS datasets. For further reference, please consult the SAS manuals.

Acknowledgements:

The author wishes to thank Ms. Peggy Alexander from the Academic Computing Department, Medical College of Virginia, for her support in preparing this document and for her assistance at the SUGI PC Hands-On Workshop. Also many thanks to Wayne Cannon, Robin Wright, and Chun Chen, whose help in the past helped make this workshop possible.

References:

*SAS is the registered trademark of SAS Institute, Inc., Cary, NC*
