Paper 105-31

A Hands-On Introduction to SAS® Basics and the SAS® Display Manager

Debbie Buck, D. B. & P. Associates, Houston, TX

ABSTRACT

This workshop is designed for SAS users who may have limited experience with the SAS Display Manager and/or the SAS DATA step. The SAS DATA step is one of the most powerful and versatile software tools available for handling and manipulating data. However, it is sometimes somewhat confusing to SAS programmers as to what the different statements do, what the correct syntax is for these statements, and how they can help you achieve your goals.

In this workshop we will examine how to get data into a SAS data set and how to manipulate data after they are in a SAS data set. We will also explore using the Display Manager, including the Enhanced Editor, Log, and Output windows, as well as the Results window, to observe the output produced, and the Explorer windows, to examine the descriptor and data values portions of the SAS data sets.

This workshop will also be helpful to SAS users who plan to attend other hands-on workshops, but are not familiar with using the SAS9 Display Manager.

INTRODUCTION

To become familiar with programming in the SAS DATA step and using the SAS Display Manager, we will discuss a number of commonly used DATA step statements including the following:

- DATA statements – How to start the creation of a SAS data set,
- INFILE/INPUT, SET, MERGE statements – How to get data into a SAS data set,
- Assignment statements - How to create or modify variables in the SAS data set,
- IF-THEN/ELSE statements - How to conditionally execute statements, and
- Subsetting IF statements - How to control which observations (records) are processed.

These statements will provide you with a basis for understanding DATA step programming. In addition, we will utilize some SAS procedures (PROC steps) and other tools in the Display Manager to help you examine your SAS data sets.

In this paper we will look at examples of a sample data set from a blood pressure medication study. The initial data include patient number, gender, date of birth, baseline blood pressure, and final blood pressure. Examples from this presentation were run on the SAS System for Windows, but the SAS code is applicable to any SAS platform.

DATA STATEMENTS

The first question to be answered is “How do I begin the DATA step?”. The DATA step always begins with a DATA statement. The purpose of the DATA statement is to tell SAS that you are starting a DATA step, to name the SAS data set(s) that you are creating, and to indicate whether this is a permanent or temporary SAS data set.

FORM: **DATA** *SAS-data-set-name*;

The data-set-name can be up to 32 characters long, is made up of letters, numbers and/or underscores, and must start with a letter or an underscore. Also, note the semi-colon at the end of the DATA statement. SAS statements all end with a semi-colon.

The following are examples of DATA statements.

```
DATA NEWPT;                ← temporary SAS data set

LIBNAME DB 'C:\SAS\MYDATA';
DATA DB.NEWPT;             ← permanent SAS data set
```

SAS data sets can be temporary or permanent. A temporary SAS data set is generally referred to with a one-level name and ceases to exist when the SAS session ends. (If running a batch program, this occurs when the job finishes executing. If in the Display Manager (interactive mode), this happens when you close the SAS session.) The SAS data set is written to the temporary WORK SAS library that is set up when the SAS session begins. The data set remains available throughout that SAS session or job. Also, even though you would refer to the temporary SAS data set above as NEWPT, the actual data set name is WORK.NEWPT.

A permanent SAS data set has a two-level name. When reading from or writing to a permanent SAS data set it is necessary to tell SAS where the SAS data set physically exists. The first level of the data set name refers to a libref – or library reference. The libref serves as a nickname or alias that points to the location of the SAS data library where the SAS data set resides. A LIBNAME statement assigns the libref to the SAS data library.

FORM: **LIBNAME** *libref 'location of directory'*;

Unlike a SAS data set name, a libref name can be only 8 characters or less. However, it is also made up of letters, numbers, and/or underscores, and must begin with a letter or an underscore. Since the libref is simply an alias, it can change from job to job while pointing to the same directory. In the permanent data set example above the LIBNAME statement associates the libref DB with the physical subdirectory 'C:\SAS\MYDATA'. The DATA statement begins a DATA step to create a SAS data set named NEWPT to be stored in 'C:\SAS\MYDATA'. The second level name NEWPT is the actual data set name.

Which statements follow the DATA statement depend upon where and in what form the data exist.

INFILE/INPUT STATEMENTS

The initial data to be imported into the SAS data set may be in a flat file, existing SAS data set(s), or a file produced by another software program. If the data is in an external file you need to tell SAS where to find the data and how to read it. What do I mean by a flat or external file? This is generally a text or ASCII file. A simple example of this might be the following layout where each record in the raw data file contains information on patient demographics and blood pressure readings.

Sample record –

```
01M 10/10/25100    90
```

Variable	Columns	Type
Patient Number	1-2	numeric
Gender	3	M, F
Date of Birth	5-12	MM/DD/YY
Baseline BP	13-15	numeric
Final BP	19-21	numeric

In order to read this external file into a SAS data set, you need two additional statements – the INFILE statement, which tells SAS where the raw data file is stored, and the INPUT statement which tells SAS how to read each record.

FORM: **INFILE** *'location of raw data file' <options>*;

The keyword INFILE tells SAS that you are working with some form of external file. The 'location of raw data file' indicates where SAS should look for the data. The INFILE statement has a number of options available which are dependent on the file structure, including specifying record length, directions on how to handle differing record lengths, and the ability to specify delimiters between variable values.

FORM: **INPUT** *record-layout*;

The information included in the INPUT statement is determined by the layout of the variable fields within the data records. The fields within a record are referred to as variables. Unlike some other software, SAS only has two kinds of variables. Character variables are text strings and can contain letters, numbers, blanks, and symbols. Variables can be standard or nonstandard. Standard numeric variables can include integers, decimal points, positive and negative numbers, and scientific (E) notation. Nonstandard numeric data include date and time values to be converted to SAS date and time values, hexadecimal and packed decimal data, and values with embedded commas and dollar signs.

INPUT statements always start with the keyword INPUT. The record layout portion of the INPUT statement is dependent upon whether the data is character or numeric and whether it is standard or nonstandard. The two most common types of INPUT statements are column input and formatted input.

Column input specifies the variable name, a dollar sign (if the variable is a character variable), and the beginning and ending columns of the variable (separated by a dash), followed by the same type of information for the next variable, continuing until each variable has been defined. Formatted input includes the beginning column using the @ column pointer, the variable name, and the informat needed to read the value, again followed by the additional variables to be read. Variable names can be up to 32 characters in length, are made up of letters, numbers, and/or underscores, and must start with a letter or an underscore.

The following DATA step creates a temporary SAS data set named NEWPT. The INFILE statement tells SAS that the raw data is located in the A drive in a file named TEST1.DAT. The INPUT statement, which in this example uses a combination of column input and formatted input, names the variables, gives the location of each variable, and tells SAS how to read each variable.

```
DATA NEWPT;
  INFILE 'A:\TEST1.DAT';          ← tells SAS where raw data exists
       INPUT @1 PTNO 2.
             @3 GENDER $1.          ← Formatted input
             @5 BDATE MMDDYY8.
                BBP 13-15           ← Column input
                FBP 19-21;
RUN;
```

The variable (field) representing the Patient Number is named PTNO. (Note: You can name variables whatever you would like as long as you follow the rules for naming variables above. It is helpful to name variables in some meaningful manner.) PTNO starts in column 1 (as denoted by the @1). The @ column pointer says that you want to start reading in column 1. The 2. informat indicates that this is a numeric variable that is 2 columns wide. Note the period after the 2 in the informat. A period is necessary with informats so that SAS knows it is an informat and not a variable name or column designation.

The GENDER variable starts in column 3 and uses the character informat $1. to indicate this variable includes text (is not numeric) and is 1 column wide. The dollar sign tells SAS that this is a character variable. The BDATE variable (Date of Birth) is a nonstandard numeric variable and requires the date informat MMDDYY8. in order to convert the field to a SAS date. SAS dates are numeric variables that are stored as the number of days since January 1, 1960. BDATE could have been read as a character

variable, but then it would only be a text string and not available for calculations or displaying in differing forms.

The final two variables BBP (Baseline Blood Pressure) and FBP (Final Blood Pressure) both use column input. BBP starts in column 13 and ends in column 15. FBP starts in column 19 and ends in column 21. SAS recognizes these as numeric variables since there is no dollar sign between the variable name and the column designation.

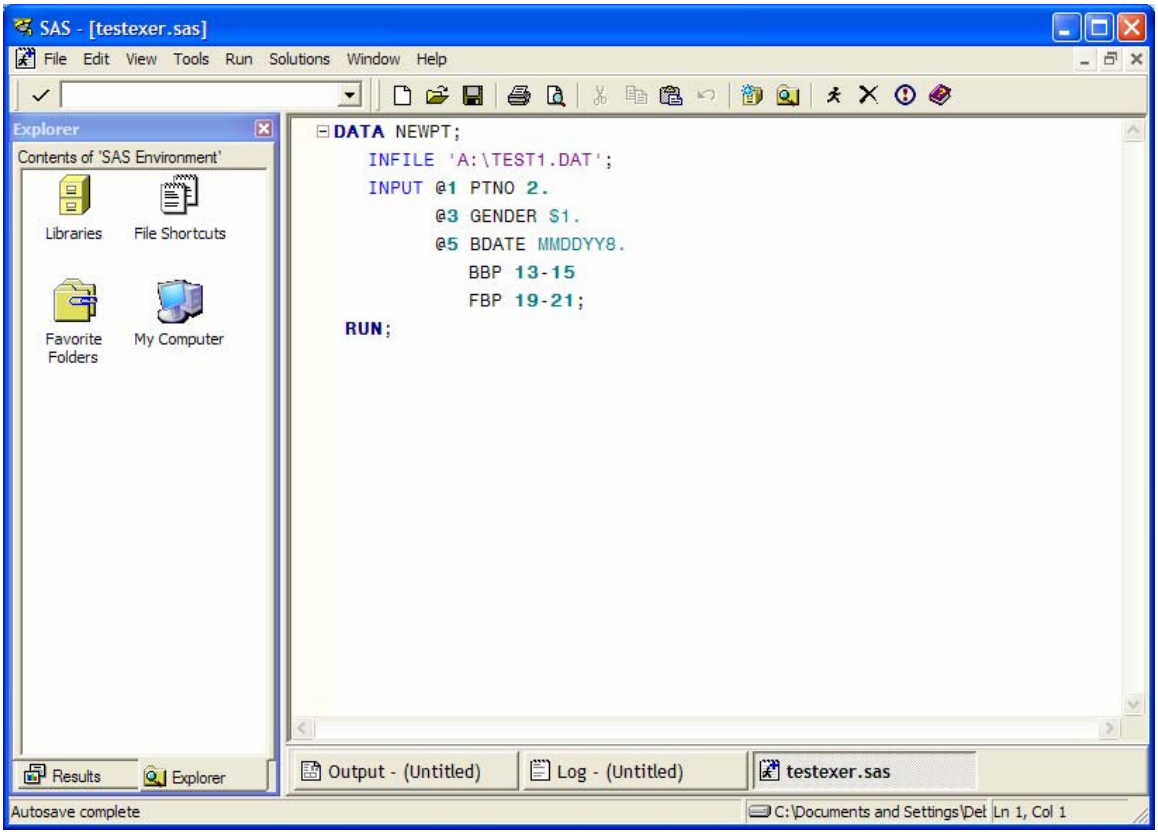
Notice the keyword RUN at the end of the DATA step. The RUN statement is used to provide a step boundary for SAS. Although a RUN statement at the end of a DATA step is not always necessary, it is a good programming habit to develop.

ENHANCED EDITOR WINDOW

Now that you are familiar with how to write the code for a basic SAS DATA step, how do you enter the SAS program code and submit the program? The SAS Display Manager allows you to enter the code and run the program, to check the log for potential problems, and to observe any results. When you click on the SAS icon on your desktop or open SAS in Windows, the Display Manager automatically opens with the Enhanced Editor as the active window.

The Enhanced Editor is very similar in appearance and editing capabilities to some other text editors, but has a number of additional features to aid you with your SAS programming. You can type in the code for the SAS program or, if the program has been written and saved, you can open the saved file using the FILE pull-down menu. Once the program has been entered, you click on the <Submit> button (which on the toolbar at the top of the Editor looks like a running person) to submit the program for execution. You can also type the command SUBMIT in the Command bar of the Display Manager.

The following shows what the Enhanced Editor looks like with the example program ready to submit. (The window has been maximized.)



LOG WINDOW

Did the program work correctly? How do you check? One of the windows in the Display Manager is the LOG window. You can easily click on the button at the bottom of the SAS screen to open the LOG window. You could also go to the VIEW pull-down menu at the top of the screen and select LOG. The LOG window then becomes the active window.

One habit you should develop is to check the SAS log any time you run a SAS program or section of a program. The SAS log will give you valuable information on the number of observations and variables, as well as any NOTEs, WARNINGs, or ERROR messages. The following is a portion of the log from our DATA step above.

Partial SAS Log

```
NOTE: The infile 'A:\TEST1.DAT' is:
      File Name=A:\TEST1.DAT,
      RECFM=V,LRECL=256

NOTE: 81 records were read from the infile 'A:\TEST1.DAT'.
      The minimum record length was 21.
      The maximum record length was 21.
NOTE: The data set WORK.NEWPT has 81 observations and 5 variables.
```

For this example, we can see that 81 records were read (which is the number we knew were in the flat file – make sure you know your data) and the NEWPT SAS data set (note that SAS refers to the data set as WORK.NEWPT) has the expected number of observations and variables. Also, there are no WARNING or ERROR messages. It is tempting to look at any resulting output first if you are working in the SAS Display Manager, since the OUTPUT window opens automatically if there is any output produced. However, make sure to always examine the log.

OUTPUT, RESULTS, AND EXPLORER WINDOWS

Every SAS data set has two parts – a descriptor portion and a data value portion. The descriptor portion is like an internal header for the data set and, among other things, contains variable names, variable types (character or numeric), and variable lengths. It also includes any labels or formats associated with the variables. (Labels and formats will be discussed later in this paper.) The descriptor and data value portions of a SAS data set can be examined using SAS procedures (PROCs) which, by default, show the procedure results in the OUTPUT window. The OUTPUT window can be accessed directly by clicking on the OUTPUT button at the bottom of the Display Manager, through the VIEW pull-down menu, or through the RESULTS window.

One way you can examine the descriptor portion of a SAS data set is with a SAS procedure (PROC). PROC steps are used to process SAS data sets. This can include producing reports or graphics, sorting data, or editing data. You can also create new SAS data sets from within a PROC step.

FORM: **PROC** *name-of-procedure* **DATA=***SAS-data-set-to-be-processed*;

PROC is the keyword to begin a PROC step. The name of procedure to be used follows the keyword PROC. After the DATA= you provide the name of the SAS data set to be processed.

PROC CONTENTS is one of SAS' many procedures. CONTENTS is very useful in examining the descriptor portion of a SAS data set. It shows details about the data set including information on the number of observations and variables and the variable names, types, and lengths. PROC CONTENTS is

particularly useful if you need to have a hard-copy of information about the SAS data set. Running the following code produces information about the descriptor portion of the NEWPT temporary SAS data set.

```
PROC CONTENTS DATA=NEWPT;
RUN;
```

Partial Output

The CONTENTS Procedure

Data Set Name:	WORK.NEWPT	Observations:	81
Member Type:	DATA	Variables:	5
Engine:	V9	Indexes:	0
Created:	Wed, Feb 01, 2006 12:52:59 PM	Observation Length:	40
Last Modified:	Wed, Feb 01, 2006 12:52:59 PM	Deleted Observations:	0
Protection:		Compressed:	NO
Data Set Type:		Sorted:	NO
Label:			

-----Alphabetic List of Variables and Attributes-----

#	Variable	Type	Len
4	BBP	Num	8
3	BDATE	Num	8
5	FBP	Num	8
2	GENDER	Char	1
1	PTNO	Num	8

To examine the data value portion of the SAS data set, you can use another procedure, PROC PRINT, to display the actual observations in the data set. By default, PROC PRINT shows you all observations and variables in the SAS data set. The following code produces output showing the data value portion of the SAS data set.

```
PROC PRINT DATA=NEWPT;
RUN;
```

Partial Output

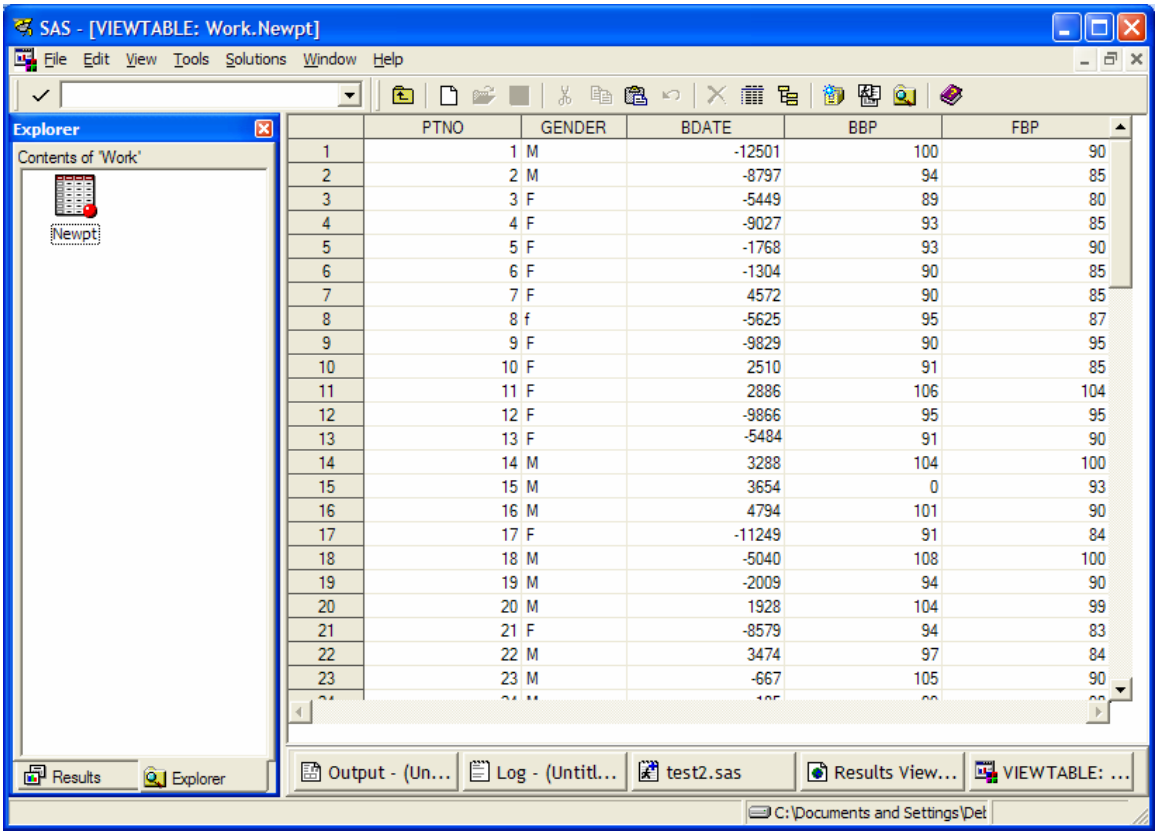
Obs	PTNO	GENDER	BDATE	BBP	FBP
1	1	M	-12501	100	90
2	2	M	-8797	94	85
3	3	F	7334	89	0
4	4	F	5583	93	85
5	5	F	-1768	93	90

As you run each PROC step the name of the procedure shows up in the RESULTS window on the left-hand side of the Display Manager. Double-clicking on the desired PROC name produces a tree-diagram type structure which allows you select what portion of the output from the PROC you would like to see displayed in the OUTPUT window. This can be very useful when you are running a number of

procedures. It also allows you to view or save only portions of procedures. Right-clicking on the PROC in the RESULTS window provides you with options such as printing, saving, or deleting that output.

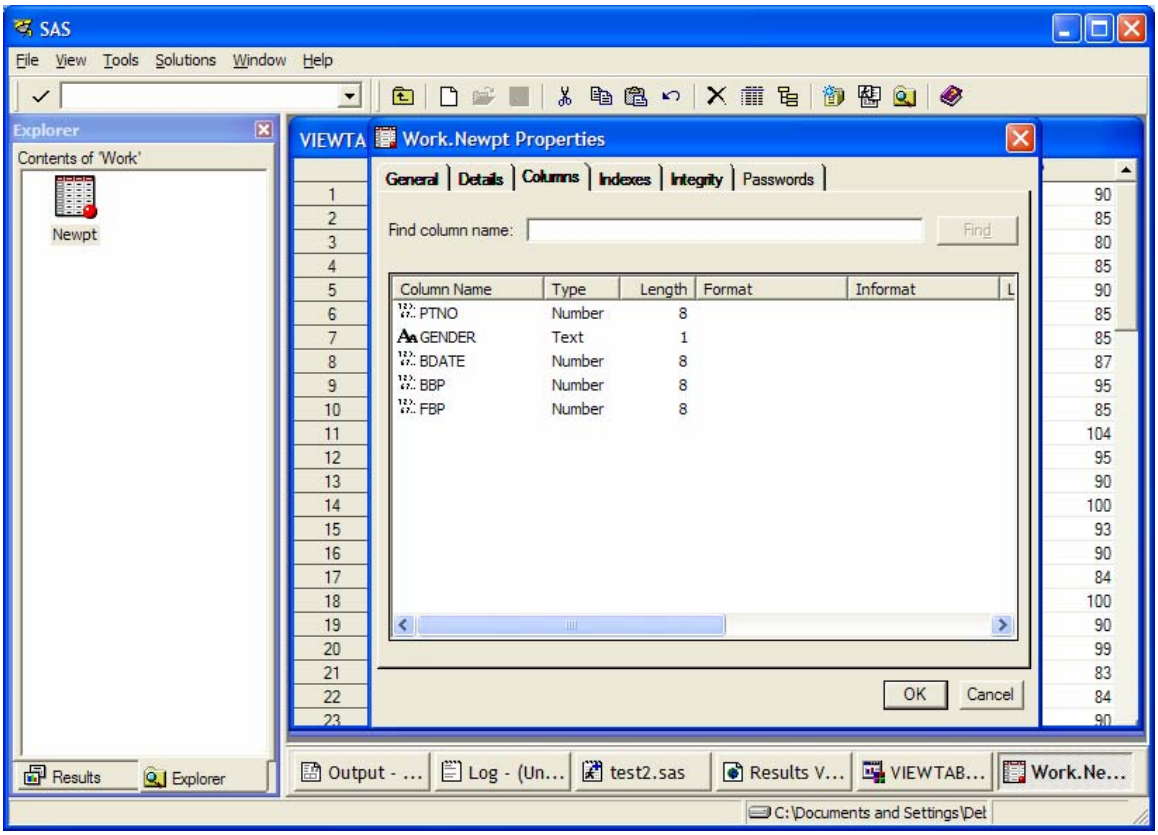
When working in the Display Manager, you can also use the EXPLORER window to examine both the descriptor and data value portions of your data sets. At the bottom of the left-hand side of the Display Manager there is a button for EXPLORER. Clicking on this will make the EXPLORER window active. In the Explorer window, if you double-click on Libraries, you will see an icon for each active library.

Since we've created a temporary SAS data set named NEWPT (which exists in the WORK SAS library), if you double-click on WORK an icon for NEWPT appears. Double-clicking opens the VIEWTABLE window which shows the data value portion of the SAS data set.



	PTNO	GENDER	BDATE	BBP	FBP
1	1	M	-12501	100	90
2	2	M	-8797	94	85
3	3	F	-5449	89	80
4	4	F	-9027	93	85
5	5	F	-1768	93	90
6	6	F	-1304	90	85
7	7	F	4572	90	85
8	8	f	-5625	95	87
9	9	F	-9829	90	95
10	10	F	2510	91	85
11	11	F	2886	106	104
12	12	F	-9866	95	95
13	13	F	-5484	91	90
14	14	M	3288	104	100
15	15	M	3654	0	93
16	16	M	4794	101	90
17	17	F	-11249	91	84
18	18	M	-5040	108	100
19	19	M	-2009	94	90
20	20	M	1928	104	99
21	21	F	-8579	94	83
22	22	M	3474	97	84
23	23	M	-667	105	90

Right-clicking on the icon causes a pull-down menu to appear. Selecting OPEN displays the VIEWTABLE window with the data value portion. Selecting VIEW COLUMNS results in a Properties window that contains variable information from the descriptor portion.



Caution: You need to remember to close the VIEWTABLE window if you want to modify the NEWPT data set or you will receive an ERROR message in the log that the data set cannot be opened.

SET, MERGE STATEMENTS

If the data you need already exist in one or more SAS data sets, then the INFILE and INPUT statements are replaced by a SET, MERGE, or UPDATE statement because SAS already knows the variable name, type of variable, and length of variable for all variables from the descriptor portion of the existing SAS data set(s). In this paper we will consider the SET and MERGE statements. Which is the appropriate statement to use?

SET Statements

If you need to bring data in from an existing SAS data set so that you can modify it, then the SET statement will, by default, bring into the new data set all variables and all observations from the existing SAS data set. The form of this DATA step is as follows.

```
DATA new-SAS-data-set;
  SET old-SAS-data-set;
RUN;
```

Although the examples in this paper use temporary SAS data sets, these could be any combination of permanent and temporary SAS data sets.

In looking at the output in the example above, we see that BDATE (Date of Birth) needs to be displayed differently to be meaningful. What you now see is the difference in the number of days between BDATE and January 1, 1960. (Thus the negative BDATE values are individuals born before January 1, 1960.) We would also like to see the BDATE, BBP, and FBP variables printed with column headers that make it

more clear as to what information these variables contain. Therefore, in our new data set we want to include a FORMAT statement and a LABEL statement.

The form of the FORMAT statement, which assigns a format to be associated with a variable (how to display the variable) is as follows.

FORM: **FORMAT** *variable-name format-name.*;

Note the period in the format-name. This is what tells SAS that it is a format, not a variable name. SAS includes a large number of informats and formats for reading and displaying variable values, including dates, times, currencies, and other types of data. You can also create your own formats using a procedure called PROC FORMAT.

The LABEL statement associates a column header for a variable. The form of a LABEL statement follows.

FORM: **LABEL** *variable-name='desired text';*

The text portion in a LABEL statement must be enclosed in quotes and can be up to 256 characters long.

When FORMAT or LABEL statements are included in the DATA step, they become a part of the descriptor and are permanently associated with the variable. You can also use FORMAT or LABEL statements in a PROC step. In this case, they only are applied to that particular PROC step.

The following code creates a new SAS data set, NEWPT_REV, that includes all variables and observations from the existing SAS data set NEWPT. It also associates a format and label with the variable BDATE and labels with BBP and FBP. (Note: In order for PROC PRINT to include the labels in the output, you need to include the LABEL option in your PROC PRINT statement.)

```
DATA NEWPT_REV;
   SET NEWPT;
     FORMAT BDATE DATE9.;
     LABEL BDATE='Date of Birth'
           BBP='Baseline Blood Pressure'
           FBP='Final Blood Pressure';
RUN;

PROC PRINT DATA=NEWPT_REV LABEL;
RUN;
```

Partial Output

Obs	PTNO	GENDER	Date of Birth	Baseline Blood Pressure	Final Blood Pressure
1	1	M	10OCT1925	100	90
2	2	M	01DEC1935	94	85
3	3	F	30JAN1980	89	0
4	4	F	15APR1975	93	85
5	5	F	28FEB1955	93	90

Do you need to concatenate (or stack) the data from two or more SAS data sets? Then the SET statement is also the appropriate statement for this situation. Again, by default, all variables and

observations in all of the SAS data sets listed in the SET statement will be included in the new SAS data set. The following is an example of concatenating two existing SAS data sets.

```
DATA NEWPT_REV;
  SET OLDPT1 OLDPT2;
RUN;
```

In this example we are creating a new data set named NEWPT_REV which contains all of the observations and variables in the existing SAS data sets named OLDPT1 and OLDPT2.

MERGE Statements

Do you need to combine (or join) two or more data sets by some common variable(s)? Then the MERGE statement is appropriate. The most common type of MERGE involves match-merging. This requires that the SAS data sets to be merged each contain the variable(s) by which you want to combine observations. It also requires that the existing SAS data sets be sorted or indexed on the variable(s). SAS data sets are sorted using the SORT procedure. PROC SORT also requires the BY statement.

FORM: **PROC SORT DATA=***SAS-data-set-to-be-sorted;*
BY *variable(s)-to-sort-by;*

The following example joins information from the NEWPT data set with the observations from the data set TREAT, matching information by the variable PTNO. Each data set must be sorted by the matching variable. NEWPT_REV will, by default, contain all variables and observations from the NEWPT and TREAT data sets.

```
PROC SORT DATA=NEWPT;
  BY PTNO;

PROC SORT DATA=TREAT;
  BY PTNO;

DATA NEWPT_REV;
   MERGE NEWPT TREAT;
    BY PTNO;
RUN;
```

Also notice that the BY statement is necessary in the DATA step to tell SAS how you want the observations joined for match-merging.

ASSIGNMENT STATEMENTS

Now that you can get the data into your new SAS data set from external data or existing SAS data sets, you need to consider whether the data needs to be manipulated in some way, such as creating or modifying variables.

If you need to create new variables, the most common way is with assignment statements. The following shows the form of an assignment statement.

FORM: *variable-name = expression;*

The expression can be a constant, mathematical operation, or a function. SAS has a large number of functions to carry out a variety of operations. The form of a function is as follows.

FORM: *function-name*(*argument1, argument2,* etc.)

The arguments for a function are dependent upon the purpose and specific requirements for a given function. Two functions that we will use in assignment statements in our DATA steps below are the TODAY and the ROUND functions. The TODAY function returns today's date (number of days since January 1, 1960). It is unusual in that it has no arguments, however, it still requires the parentheses to indicate that it is a function. The ROUND function has two arguments. The first argument is the variable to be rounded off. The second argument tells SAS what units should be used for rounding.

For example, if you need to create a variable named AGE, based on the current date, you can use an assignment statement with a SAS date function and a mathematical operation in the DATA step.

```
DATA NEWPT_REV;
  SET NEWPT;
    AGE = (TODAY() - BDATE)/365;
RUN;
```

You can also use assignment statements to modify existing variables. The following shows an example of modifying an existing variable. You could also have created a new variable in the second assignment statement. In the example below the variable AGE is rounded off to whole integers.

```
DATA NEWPT_REV;
  SET NEW;
    AGE = (TODAY() - BDATE)/365;
     AGE = ROUND(AGE,1);
RUN;
```

IF-THEN/ELSE STATEMENTS

IF-THEN statements can be used to conditionally create or modify data.

FORM: **IF** *expression* **THEN** *statement*;
ELSE *statement*;

The IF expression is some condition that must be satisfied as true in order for the operation (statement) following the THEN to be carried out. If the expression is true, then SAS carries out the operation in the THEN statement, and skips over the ELSE statement. If the expression is false, then SAS does not carry out the operation in the THEN statement, but goes to the ELSE statement. (Note: It is not necessary to include an ELSE statement with an IF-THEN statement.)

For example, in this case you need to create a new variable to identify at which medical center a patient was enrolled, based on patient number. Patients 1-30 were enrolled at Center 1, patients 31-60 were enrolled at Center 2, and patient numbers 61 and above were enrolled at Center 3. The following IF-THEN/ELSE statements will allow you to conditionally assign values to your new variable, CENTER.

```
DATA NEWPT_REV;
  SET NEWPT;
   AGE=ROUND(((TODAY()-BDATE)/365),1);
    IF PTNO LE 30 THEN CENTER=1;
      ELSE IF PTNO GT 30 AND PTNO LE 60 THEN CENTER=2;
        ELSE IF PTNO GT 60 THEN CENTER=3;
RUN;
```

SUBSETTING IF STATEMENTS

If you need to subset your data based upon the values of one or more variables, you can use a Subsetting IF statement.

FORM: **IF** *expression;*

The expression is evaluated, and if the expression is true, processing continues on that observation. If the expression is false, processing on that observation stops, the observation is not saved to the data set being created, and the DATA step moves on to the next observation.

The following example limits the observations in the data set to those patients from Center 1 who are under 30 years of age. Note that the expression can be a compound expression using the operators AND or OR (among others).

```
DATA NEWPT_REV;
    SET NEWPT;
     AGE=ROUND(((TODAY()-BDATE)/365),1);
       IF PTNO LE 30 THEN CENTER=1;
         ELSE IF PTNO GT 30 AND PTNO LE 60 THEN CENTER=2;
          ELSE IF PTNO GT 60 THEN CENTER=3;
     IF CENTER=1 AND AGE LT 30;
 RUN;
```

CONCLUSION

In this workshop we have examined how to get data into a SAS data set, how to create new variables or modify existing variables, how to conditionally handle variables or observations, and how to control which observations are written to the SAS data set. Each of the statements we have examined has additional capabilities which were beyond the scope of this presentation. We also used the SAS Display Manager to submit our SAS job, check the log, and examine the descriptor and data value portions of the SAS data set, as well as any output produced. Hopefully, this paper has provided you with guidelines to make SAS DATA step programming and using the SAS Display Manager user-friendly to you.

REFERENCES

SAS Institute Inc. (1990) *SAS Language and Procedures Guide, Version 6, First Edition*, Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1999) *SAS Online Doc., Version 8*, Cary, NC: SAS Institute Inc.

SAS Institute Inc. (2005) *SAS Online Doc., Version 9.1.3*, Cary, NC: SAS Institute Inc.

AUTHOR CONTACT INFORMATION

Debbie Buck
D. B. & P. Associates
Houston, TX 77095
Voice: 281-256-1619
Fax: 281-256-1634
Email: debbiebuck@houston.rr.com