# SAS Formats – Simply Powerful

by

Howard Hagemann
Program Specialist
Student Assessment Division
Texas Education Agency

South Central SAS User's Group Conference

October 2007

# SAS Formats – Simply Powerful

by Howard Hagemann

## Abstract

SAS formats are powerful and simple to use. In this paper I will discuss the creation of formats using PROC FORMAT, using formats to identify groups and subgroups, using SAS modifiers to indicate special instructions and using SAS informats to read raw data. SAS formats make easy work of complex tasks. They make output easier to read and interpret.

## Introduction

This paper is written to cover multiple uses of SAS formats. The power of PROC FORMAT to transform data will be discussed in detail. The uses of informats and SAS modifiers to read data values will be explored.

The ability to transform data by assigning descriptive labels to data values that can be recognized for future reference is rudimentary to SAS formats. Having procedures that can recognize the descriptive labels assigned to the original values for data manipulation provides a powerful tool for users. In this paper I will discuss syntax and give examples of how formats can transform data and how to utilize the results.

Getting the data into SAS is essential to the process. SAS informats provide a means to read data of various formats into SAS. SAS modifiers are an extension of the SAS informat. Syntax and examples of SAS informats and SAS modifiers will be discussed in detail.

## The FORMAT Procedure – VALUE Statement

PROC FORMAT is the SAS procedure that is used to build user defined formats. Suppose you have data for student scores on exams in biology. You want to break the scores into grades based on the normal levels where an “A” is for scores 90 to 100, a “B” is 80 to 89, a “C” is 70 to 79, a “D” is 60 to 69 and an “F” is below 60. The syntax for the PROC FORMAT procedure to format the scores into the various grades would be as follows:

```
 proc format;
   value grade   0  - 59    = ‘F’
                60 - 69   = ‘D’
                70 - 79   = ‘C’
                80 - 89   = ‘B’
                90 – 100  = ‘A’
              ;
```

The above code for the format procedure contains the PROC FORMAT statement. A value statement that contains the name for the format named grade is given. The ranges of data are assigned a value as “A”, “B”, “C”, etc. Notice that the semi-colon is not used until after the last range is listed.

The example below uses the format created in the above FORMAT procedure to format the output of the numeric variable score to the corresponding grade. Notice the FORMAT statement that was used to invoke the user defined format grade.

```
data stu;
  input name & $15. score ;
  datalines;
Mary Jones      82
Jeff Smith      61
Sue Austin      93
Fred Hart      100
  ;
  proc print data=stu;
  format score grade.;               /*  format statement   */
  run;
```

The output from the above example is shown below.

```
Obs       name        score

 1     Mary Jones       B
 2     Jeff Smith       A
 3     Sue Austin       A
 4     Fred Hart        A
```

A value range can specify

- a single value such as 33 or ‘T’
- a range of numeric values, such as 0 – 500
- a range of character values enclosed in quotation marks, such as ‘A’ – ‘M’
- a list of unique values separated by commas, such as 90, 180, 270 or ‘B’, ‘D’, ‘F’.

These values can be character or numeric, but not a combination of both because the FORMAT procedure must be either character or numeric.

When the specified values are character values, they must be enclosed in quotation marks. The format’s name must start with a dollar sign ($). When the specified values are numeric values, they are not enclosed in quotation marks and the format name should not begin with a dollar sign.

You can also use the keywords LOW and HIGH to specify the lower and upper limits of a variables range. The keyword LOW does not include missing values but does include missing character values. The keyword OTHER can be used to label missing values as well as any values that are not specifically addressed in the range.

When specifying a label for displaying each range,

- enclose the label in quotation marks
- limit the label to 256 characters
- use double quotation marks if you want an apostrophe to appear in the label.

Date ranges are special because they have special formatting needs to get the values into a date format.  Shown below is a VALUE statement with ranges created by date values.

```
Value dates
  ‘01jan2000’d  -  ‘31mar2000’d  =  ‘1st quarter’
  ‘01apr2000’d  -  ‘30jun2000’d  =  ‘2nd quarter’
  ‘01jul2000’d  -  ‘30sep2000’d  =  ‘3rd quarter’
  ‘01oct2000’d  -  ‘31dec2000’d  =  ‘4th quarter’
   ;
```

Notice that each date is enclosed in single quotes and followed by the letter ‘d’.

When creating VALUE statements, I like to put the semi-colon on a separate line so that it is obvious.  It is easy to overlook.

**The FORMAT Procedure – PICTURE Statement**
Another statement that can be used to create user defined output is the PICTURE statement.  PICTURE formats are different from value formats in two respects. PICTURE formats can be used with numeric variables only.  The PICTURE format creates a template that is used to display numeric variable values.

PICTURE statements can be specified in two different ways.
1) digit selectors
2) message characters

Digit selectors are numeric characters (0 through 9) that define positions for numeric values.  If you use nonzero digit selectors, zeros are added to the formatted value as needed.  If you use zeros as digit selectors, no zeros are added to the formatted value.  If the PICTURE statement shows output of ‘99’ then the result of a value 01 would be 01. If a picture output value is shown as ‘00’ then the output value would be 1.

The PICTURE statement below uses ‘099’ and ‘999’ as the template for printing the numbers and a prefix option to add the alphabetic grade to the printed output.

```
proc format;
   picture grade  (default = 10)
  low - 59   = '099' (prefix = 'F  -  ')
   60 - 69   = '099' (prefix = 'D  -  ')
   70 - 79   = '099' (prefix = 'C  -  ')
   80 - 89   = '099' (prefix = 'B  -  ')
   90 - 99   = '099' (prefix = 'A  -  ')
   100       = '999' (prefix = 'A+ - ')
                 ;
```

The results of the above PICTURE statement are shown below.

```
Obs       name        score

 1     Mary Jones    B  -  82
 2     Jeff Smith    D  -  61
 3     Sue Austin    A  -  93
 4     Fred Hart     A+ - 100
```

When using PICTURE formats, the templates specify how numbers are displayed and provide a method for prefixes, leading zeros, decimal and comma placement, embedding characters within numbers, truncation and rounding of numbers.

Message characters are non-numeric characters that can be specified. They are inserted into the picture after the numeric digits are formatted. An example of a message statement would be the following:

```
picture day 01-31 =  '99'
            Other = '99 – Illegal Day Value'
             ;
```

For the values 02 and 67 the result would appear as follows:

```
                     02
67 – Illegal Day Value
```

Ranges can be specified with special keywords like LOW, HIGH, and OTHER. The keyword LOW includes missing values for character formats and does not include missing values for numeric formats.

A list of options that can be used in the PICTURE statement are shown below.

- FILL = ‘character’ specifies a fill character. This character replaces the leading characters of the picture until a significant digit is encountered. The default FILL = ‘ ‘ (blank).
- PREFIX = ‘character’ is a one or two character prefix placed in front of the first significant digit of the value.
- MULTIPLIER = n specifies a number to be multiplied by a value before it is formatted. The main purpose of the MULT=option is to get a data value containing decimals into a form that will fit into the picture template.

**Format Options for VALUE or PICTURE Statements**

Below is a list of format options that are common to both the VALUE statement and the PICTURE statement.

- MIN = n specifies a minimum width for the format.
- MAX = n specifies a maximum width for the format.
- Default = n specifies the default width if the format does not have a width specification.

- FUZZ = n specifies a fuzz factor. If a number does not match a value or fall within a range exactly, but comes within the fuzz value, the FUZZ factor is used.

**The FORMAT Statement**
When reviewing the above code for either the VALUE statement or the PICTURE statement, it was a FORMAT statement that was used to invoke the user defined format. The FORMAT statement syntax is shown below.

```
  format variable(s) [format.];
```

Notice that multiple variables can be associated with a format. Also, recognize that a period is placed at the end of the format name. The format can be a user defined format created by the PROC FORMAT procedure or can be an internal SAS format.

**The ATTRIB Statement**
Another statement that can be used to invoke a format is the ATTRIB statement. The general form of the ATTRIB statement is as follows:

```
ATTRIB variable [FORMAT=format] [INFORMAT=informat] [LABEL='label']
[LENGTH=[$]length];
```

In the above program that uses the PICTURE statement, an ATTRIB statement that could replace the FORMAT statement is shown below.

```
  attrib score format=grade. Label='RESULTS';
```

**Using SAS Informats & Modifiers**
A variable's informat is the pattern that SAS uses to read raw data values into a variable. The default informat is Best12. for numeric variables and $w. for character variables. If a character value ($) is used in list input then the default width for a character input value is a maximum of eight bytes wide. User defined formats created by the FORMAT procedure can be used to read raw input data. When creating a user defined informat the INVALUE statement is used instead of the VALUE statement.

```
proc format;
  invalue grade   0  - 59    = 'F'       /*  invalue statement  */
                 60 - 69     = 'D'
                 70 - 79     = 'C'
                 80 - 89     = 'B'
                 90 - 100    = 'A'
               ;
```

SAS modifiers can be used with list input data. The two format modifiers are the colon and the ampersand. The colon allows the user to specify an informat for reading nonstandard data values. The ampersand gives the user the ability to specify an informat for reading nonstandard data values that contain single embedded blanks.  An example of reading raw data using SAS modifiers is given below.

```
data mod1;
  input city & $15. count : comma9.;
  datalines;
Denver         123,456,789
Austin             333,333
San Antonio    715,666,333
  ;
  proc print data=mod1;
  run;
```

```
Below is the output from the above program.  Notice the ampersand after
the input variable named city and the colon followed by "comma9." after
the variable named count.
```

```
                                    Obs    city                count

                                     1     Denver          123456789
                                     2     Austin             333333
                                     3     San Antonio     715666333
```

**Creating New Variables with the PUT and INPUT Function**
In many applications you must convert one data type to another.  You may want to transform digits that are a character string to a numeric value.  You may want to covert a numeric value to a character string.  It can be done implicitly by forcing SAS to do the conversion for you or explicitly with the input and put functions.  The INPUT function converts a character value to a numeric value and the PUT function converts a numeric value to character value.  User defined formats as well as SAS internal formats can be used to produce the desired output.

The syntax for the INPUT function is as follows:

INPUT(argument, informat);

where "argument" is a character constant, variable or expression and "informat" is a SAS informat that is usually numeric.

The syntax for the PUT function is shown below.

PUT(argument, informat);

Below is an example of the PUT function creating a character variable named "result"  by converting numeric data to character data where score is a numeric variable.

```
options fmtsearch=forlib;

data stu;
  input name & $15. score ;
  result = put(score, grade.);    /*  put statement  */
  datalines;

Mary Jones      82
Jeff Smith      61
Sue Austin      93
Fred Hart      100
  ;

  proc print data=stu;
   run;
```

In this example a stored format "grade" was called rather than recreating a user defined format.

The output from the above program is shown below.

```
Obs       name        score    result

 1     Mary Jones      82        B
 2     Jeff Smith      61        D
 3     Sue Austin      93        A
 4     Fred Hart      100        A
```

**Creating Permanent User Defined Formats**

When you create a permanent format to associate with a variable in a data set, be sure that the permanent format is always available in the assigned library.  Always include a LIBNAME statement referencing the permanent format in the program that references the format.  The permanent stored format can be created with either a PICTURE statement or a VALUE statement.  The following code creates a permanent user defined format named grade in a 'SAS-data-library'.

libname library 'SAS-data-library';

```
proc format library=library;
proc format;
   picture grade  (default = 10)
  low - 59   = '099' (prefix = 'F  -  ')
   60 - 69   = '099' (prefix = 'D  -  ')
   70 - 79   = '099' (prefix = 'C  -  ')
   80 - 89   = '099' (prefix = 'B  -  ')
   90 - 99   = '099' (prefix = 'A  -  ')
   100       = '999' (prefix = 'A+ -  ')
                 ;
run;
```

To retrieve a format stored in a catalog other than work.formats (temporary) use the options statement as shown below.

options fmtsearch=*library*;

The library where the permanent format was stored that is given in the above options statement is used to find the user defined format shown in this PROC PRINT statement.

```
proc print data=sample;
format scores grade.;
run;
```

**Using FMTLIB with PROC FORMAT to Document Formats**

When you have created a large number of permanent formats, it can be easy to forget the exact spelling of a specific format name or its range of values. The PROC FORMAT statement displays a list of all the formats in the specified catalog when the keyword FMTLIB is added to the end of the statement. Below is an example of using the FMTLIB keyword at the end of the PROC FORMAT statement.

```
libname library ‘c:\sas’;
proc format lib=library fmtlib;
run;
```

The output is shown below.

```
|      FORMAT NAME: GRADE    LENGTH:    1   NUMBER OF VALUES:    5
|   MIN LENGTH:   1  MAX LENGTH:  40  DEFAULT LENGTH   1  FUZZ: STD
|
|START              |END                |LABEL  (VER. V7|V8   12JUN2007:14:27:11)|
|                  0|                 59|F
|                 60|                 69|D
|                 70|                 79|C
|                 80|                 89|B
|                 90|                100|A
```

In the above example the only format that was in the catalog was the format named ‘grade’.

The SELECT and the EXCLUDE statements can also be used to process specific formats rather than an entire catalog;

```
PROC FORMAT LIB=library FMTLIB;
   SELECT format-name;
   EXCLUDE format-name;
RUN;
```

**Formats – Simply Powerful**
Having the ability to change data using descriptive labels and then having procedures that can recognize the descriptive labels for processing makes the SAS format process a powerful tool. Having the ability to create user defined formats gives the process the flexibility needed to handle many complex jobs. Knowing that the PROC steps will recognize the descriptive labels allows users to transform data into the results they need. In this section some examples of the power of formats will be given.

The simple scores to grades example that was used in earlier sections of the paper demonstrates how the format procedure can give added value to data. The data is shown below.

```
Mary Jones      82
Jeff Smith      88
Sue Austin      93
Fred Hart       75
Sally Smith     66
Judy Karnes     99
Lana Parks      87
```

Counts and percentages given by the PROC FREQ procedure are often useful in reviewing test results. The following tables are created by PROC FREQ using the data above.

```
                          The FREQ Procedure

                                             Cumulative    Cumulative
score      Frequency      Percent     Frequency       Percent
----------------------------------------------------------------
    D              1        14.29              1         14.29
    C              1        14.29              2         28.57
    B              3        42.86              5         71.43
    A              2        28.57              7        100.00
```

The FREQ procedure provides statistics on the distributions of the grades derived from the score data. The code for the FREQ procedure is listed below.

```
proc freq data=stu;
  table score;
  format score grade.;         /* format statement */
  run;
```

The code is simple but it gives useful results. The real power in the PROC FREQ procedure is the FORMAT statement. We know the number of students and the percentages at each assigned grade level. You can see from the output that the class did well. Formatting gives new meaning to the data.

The format procedure can be a useful tool for finding erroneous data. For instance, the price of items in a department store would normally run from five dollars to two hundred dollars. You want to identify any items that are priced outside this range. A user defined format can identify all items within the range or outside the range. A FORMAT procedure that would work for this process is shown below.

```
 proc format;
    value prices  5 – 200  =  ‘within’
   other                   =  ‘outside’
;
```

In the print procedure or in a data step you would invoke the format using the syntax shown below.

```
 format price  prices.;
```

Any item where price has the descriptive label of ‘outside’ could be printed to identify data outside the normal range. One way to capture the unknown is to first define the known elements.

**Conclusion**

Getting data into SAS is obviously essential to the process. SAS informats provide a means to read data of various formats into SAS. SAS modifiers are an extension of the SAS informat. They provide additional flexibility for reading data into SAS.

The ability to transform data by assigning descriptive labels to data values that can be recognized for future reference is rudimentary to SAS formats. The VALUE statement brings flexibility and readability to data when descriptive labels can be assigned. Having procedures that can recognize the descriptive labels assigned to the original values for data manipulation provides a powerful tool for users. This was exemplified by the raw scores to grades frequency distribution.

The PICTURE statement provides a means for putting numeric output into a useful format by adding prefixes, leading zeros, decimal placement, comma placement, embedding characters within numbers, truncation and rounding of numbers. When numeric data is being used, a PICTURE statement can bring meaning to variables being output.

SAS efficiencies are gained when formats can be used to accomplish tasks. The FORMAT procedure uses a binary search using the lookup table that makes efficient use of computer resources. Centralized maintenance is another benefit of using the FORMAT procedure. Performance efficiencies and the ease of use make SAS formats a viable solution for SAS users.

**References**

SAS Online Documentation. Version 9.1, Cary, NC: SAS Institute, SAS Institute, Inc.
SAS Language and Procedures

Carpenter, Arthur L. “Building and Using User Define Formats”. Proceedings of the Twenty-ninth annual SAS Users Group International Conference. Cary, NC: SAS Institute, Inc.

**Acknowledgements**
I would like to acknowledge the Analysis and Reporting Unit in the Student Assessment Division of the Texas Education Agency for their support. I would like to thank Jenny Eaton, Joel Bourgeois and Stephanie Weaver for reviewing the paper.

**Contact Information**

Howard Hagemann
Program Specialist
Student Assessment Division
Texas Education Agency
Austin, TX
Work phone: (512) 463-2597
Email: howard.hagemann@tea.state.tx.us