Formats: A Beginner's Tutorial

Richard D. Langston, SAS Institute Inc.

ABSTRACT

This tutorial will introduce the beginning user of the SAS® System to the concept of formats. Several Institute-supplied formats will be described so that the user can better understand the concept. Then, PROC FORMAT will be introduced as the method by which the user can produce custom formats. Various features of PROC FORMAT are discussed, including ranges, exclusion ranges, LOW/HIGH/OTHER, informats, PICTURE, and the CNTLIN= data set.

INTRODUCTION

For any value that you wish to print when using the SAS System, you will use a format. A format is simply a method for converting data from one representation to another. There are many standard formats that are supplied as part of the SAS System, and you can extend that list by creating your own custom formats. In this tutorial, you will learn the basics about how the SAS System uses formats to produce useful output. You will see several Institute-supplied formats used as examples. And you will learn how to produce your own formats for your own particular needs.

The SAS System deals with numeric and character data within its SAS data sets. All numeric data are stored in floating point representation, a binary form of what is commonly known as scientific notation. The representation is necessary for a computer operating system to perform numeric calculation on the value, but the representation is very difficult for humans to handle.

For example, consider the value 1 on the IBM mainframes. The floating point representation using hexidecimal notation is 4110000000000000. And on PCs and on most UNIX boxes, the value is 3FF0000000000000. This value is certainly not intuitively the value 1 to us humans.

For this reason, the SAS System provides a method to represent the arcane floating point data in a form that is easily recognized. This method is the format.

THE W.D AND BEST FORMATS

Probably the simplest format is the 'w.d' format, where w refers to a number of columns and d refers to the number of columns devoted to digits to the right of a decimal point. So consider the following values and how the w.d format would be used to produce readable versions of the values (note that 'b' is used to denote a blank):

value	format 5.2	format 5.1	format 5.0
1	b1.00	bb1.0	bbbb1
1.5	b1.50	bb1.5	bbbb1
1.6	b1.60	bb1.6	bbbb2
1.63	b1.63	bb1.6	bbbb2
1.99	b1.99	bb2.0	bbbb2
100000	1E5	1E5	1E5

Several points should be made about the above examples. Note that all the values in the various columns line up by the decimals (except for the 100000 value, which is explained below). This is because most reports require this, so that it is easier to read and natural to the eye. Note also that rounding will occur if the value cannot be fully represented in the space alloted. For example, the format 5.0 cannot represent 1.6 exactly, because it doesn't allow for any digits to the right of the decimal. Therefore, the 5.0 format rounds the 1.6 value to a 2. Note also that 1.5 rounds up. In the case of 100000, the value cannot fit in the allowed space of 5 columns even with rounding, because the number requires 6 columns. When this happens, w.d reverts to another format called BEST. BEST will determine the best representation for the number given the number of columns available. In our example, BEST chooses to use exponential notation in formatting the value.

REPORT-ORIENTED FORMATS

Now that we've seen how w.d and BEST process numeric values, let's look at another set of formats. These are more specialized forms of numeric formats that can be used in reports. These formats are COMMA, DOLLAR, NEGPAREN, and E. Here is a small DATA step that demonstrates these formats:

```
data _null_; file print header=h;
     input x @@;
     put x 7.2 +1 x comma8.2 +1 x dollar9.2
               +1 x negparen9.2 +1 x 10.2;
     return;
h:   put +7 +1 '   comma' +1 '   dollar' +1
     ' negparen' +1 '          e';
     cards;
1 10 100.35 1234.56
.25 -.25 -25
;
```

	comma	dollar	negparen	e
1.00	1.00	$1.00	1.00	1.000E+00
10.00	10.00	$10.00	10.00	1.000E+01
100.35	100.35	$100.35	100.35	1.004E+02
1234.56	1,234.56	$1,234.56	1,234.56	1.235E+03
0.25	0.25	$0.25	0.25	2.500E-01
-0.25	-0.25	$-0.25	(0.25)	-2.500E-01
-25.00	-25.00	$-25.00	(25.00)	-2.500E+01

As with w.d, the formats described above are intended to be used with columnar reports. The COMMA and DOLLAR formats put commas between each 3 digits. The DOLLAR format also adds a leading dollar sign. The NEGPAREN format uses parentheses to indicate negative numbers, a popular representation in accounting.

DATE FORMATS

Another group of Institute-supplied formats is the date processing group. Since this group of formats is discussed in more detail in another tutorial, they will only be briefly discussed here.

A SAS date value is actually a day offset from January 1, 1960. The offset can be negative or positive, depending on whether the date is prior to January 1, 1960. For example, my birthday of March 19, 1955 has a SAS date value of -1749, and the opening day of WUSS, October 28, 1994 has a SAS date value of 12717. As with floating point binary numbers, these SAS date values are somewhat meaningless to us humans unless they can be represented in a more useful form. The example below demonstrates several date formats.

```
data _null_;
     input x @@;
     put 'date:' x date7. +1 x date9.;
     put 'weekdate:' x weekdate.;
     put 'mmddyy:' x mmddyy6. +1 x mmddyy8.;
     put 'julian:' x julian5. +1 x julian7.;
     cards;
1 10 100.35 1234.56
run;

date:19MAR55 19MAR1955
weekdate:      Saturday, March 19, 1955
mmddyy:031955 03/19/55
julian:55078 1955078
date:26OCT94 26OCT1994
weekdate:  Wednesday, October 26, 1994
mmddyy:102694 10/26/94
julian:94299 1994299
```

The above example shows different ways that the date values are represented. It also shows how different width specifications can change the looks of the formatted value. As with our w.d examples above when rounding took place, the various date processing formats will produce different results when different widths are given. For example, DATE7. formats -1749 as 19MAR55 while DATE9. formats it as 19MAR1955, allowing for a four-digit year.

The use of WEEKDATE without a width demonstrates the concept of "default width". When the format was implemented at SAS Institute, it was decided how many characters were appropriate for formatting if the user didn't provide a width. It was decided that 29 was an appropriate number (length('September')+length('Wednesday')+ 11 for all other characters), so that number is used whenever the user does not supply a width. It is good practice to always use a width specification, especially when producing reports. This is because you can be explicit about the widths you expect for the various columns of output.

INTRODUCING PROC FORMAT

Now that you have seen how Institute-supplied formats function within the SAS System, let us consider those formats that you can generate yourself.

As an example, consider some demographic data that contains a person's specific age, but the report you want is to list data in terms of demographic groups. An example of a DATA step reading in such data:

```
data demo;
     input name $ age sex $ income; cards;
John 40 M 3
Mary 35 F 3
Bill 24 M 2
Jane 26 F 2
run;
```

In the above DATA step, the continous value of age is put into the variable AGE, and the variable INCOME is set to an income category, which is either 2 or 3 based on the above data.

In the report, you want to group certain ages into certain demographic groups, and you want to properly identify the income groupings. You can use the following PROC FORMAT statements to create formats to handle the groupings:

```
proc format;
value age
     low-17  = 'child'
     18-34   = 'young adult'
     35-49   = 'adult'
     50-64   = 'older adult'
     65-high = 'senior';
value incomec
     1       = '<20K'
     2       = '20K-30K'
     3       = '30K-40K'
     4       = '40K-50K'
     5       = '50K+';
run;
```

In the above example, we divide age into 5 categories, and give each category a meaningful label. For income, we provide useful labels for the income categories. The keyword LOW refers to any value, negative or positive, below the end range value of 17. The keyword HIGH refers to any value greater than the start range value of 65.

What the above example has produced is two new formats called AGE and INCOMEC. These formats can be used the same way that Institute- supplied formats are used to translate numeric data into more recognizable character data. Consider our data set called DEMO that was created above:

```
proc print data=demo;
     format age age. income incomec.;
     run;
```

We have associated the formats we've created with the appropriate variables. Note that the format name AGE does happen to have the same name as the variable AGE, but this is certainly not required. The format INCOMEC does not have the same name as the corresponding variable INCOME. Also, the same format can be applied to many different variables in the data set. In our example, there is only one variable to which to apply each format. Here is the output produced by the PROC PRINT step described above:

OBS	NAME	AGE	SEX	INCOME
1	John	adult	M	30K-40K
2	Mary	adult	F	30K-40K
3	Bill	young adult	M	20K-30K
4	Jane	young adult	F	20K-30K

CHARACTER FORMATS

Although our discussion has only mentioned numeric formatting, it is possible to format character data as well. This is not always as common, since character data may already be adequately descriptive and will not need to be relabeled. However, consider an example involving grade letters:

```
proc format;
     value $grade
          'A'='superior'
          'B'='above average'
          'C'='average'
          'D'='below average'
          'E','F'='failing'
          other='(unknown)';
```

A character format is indicated by placing a $ before the remaining name of the format. Note also in the above example that multiple ranges can have the same label. Both 'E' and 'F' are translated to the label 'failing'. Another feature seen here is the

OTHER keyword. If a value to be formatted cannot be matched with any other specified ranges (in the case above, anything other than A, B, C, D, E, or F), the label associated with the OTHER clause will be used for formatting.

NORMAL AND EXCLUSION RANGES

In the examples above, we have seen both single values and value ranges corresponding to labels. An example of a single value range is

```
'A'='superior'
```

and an example of a normal range is

```
35-49='adult'
```

Suppose that instead of dealing with ordinal values like age, we have continuous data that are mathematically derived. We still want specific cutoffs, but we want the cutoff value in only one category, since overlapping ranges are not permitted. Consider this format definition:

```
proc format;
     value rr
         1-<10='less than 10'
         10-20='10 through 20'
         20<-30='above 20 and up to 30';
```

In this example, the < symbol indicates that a value is being excluded from the range. The range 1-<10 means that anything between 1 and 10,except for the value of exactly 10, will be formatted as 'less than 10'. The < symbol comes after the start range value and before the end range value.

```
data _null_;
     input x @@;
     put x= x=cats.;
     cards;
1 9.999 10 15 20 25
run;

X=1 X=less than 10
X=9.999 X=less than 10
X=10 X=10 through 20
X=15 X=10 through 20
X=20 X=10 through 20
X=25 X=above 20 and up to 30
```

PICTURE FORMATS

There is a specialized type of numeric format that can be created by PROC FORMAT. This format is used to represent numeric values with special characters inserted to give the number a more recognizable meaning. For example, consider the Institute's phone number, 9196778000. This number is somewhat difficult to read as a stream of digits. But then consider it using a form that you're probably used to seeing: 919/677-8000. Now the number is easy to recognize and read. The numeric value for the phone number can be mapped into this configuration by providing a series of digits called "digit selectors". Here is the PICTURE statement to produce the phone number as it has been presented:

```
proc format;
     picture phone other='999/999-9999';
     run;
```

When our number 9196778000 is formatted, the first three digits match with the first three digit selectors in the picture. Then the / will be placed in the formatted value. The next three digits of our number will be mapped into the formatted value, followed then by the dash that appears in the picture statement. The last four digits of our number then get mapped into the formatted value. Note that because we used OTHER above, all numeric values will be mapped using this method.

A digit selector value of 9 means that a digit will always be formatted. A digit selector of 0 means that a leading zero digit will be formatted as a blank, but any other digit will be formatted. This feature exists to allow for standardized columnar formatting. Consider this picture:

```
proc format;
     picture nolead other='009,999.99';
     picture lead other='999,999.99';
     run;
data temp; input x @@; y=x; cards;
1000 10000 100000
run;
proc print; format x lead. y nolead.; run;
```

OBS	X	Y
1	001,000.00	1,000.00
2	010,000.00	10,000.00
3	100,000.00	100,000.00

The difference in the performance of the two pictures is that a leading zero will not be seen with a 0 digit selector. You will need to decide whether leading zeroes are appropriate for your report.

CNTLIN= DATA SET

So far, all examples have been shown using the VALUE and PICTURE statements. You may find that you have data that properly indicates what the ranges and labels are supposed to be, but you don't want to type in all the VALUE statements yourself, or write a SAS program to convert the file into a series of VALUE statements. As an alternative, you can provide format definitions to the FORMAT procedure by using the CNTLIN= option. This option allows you to indicate the name of a SAS data set to contain all the information that could be supplied on a VALUE statement. The data set is expected to contain certain variable names. The minimum variables to appear are FMTNAME, a character variable containing the name of the format; START, the value of the starting point of a range; and LABEL, the label value. There should be one observation per range per format. Consider this example:

```
data cntlin;
     fmtname='demo';
     input @1 start $ @5 end $
           @10 label $char20.;
     cards;
low  17    child
18   34    young adult
35   49    adult
50   64    older adult
65   high  senior
run;
proc format cntlin=cntlin;
run;
```

This will produce the same format as the one created by the VALUE statement for DEMO earlier in this tutorial.

Note that START and END are created as character variables to

allow for the character strings 'low' and 'high' to appear in the data.

SUMMARY

Formats allow for you to present your data in a form that will be most easily recognized by the user of your report. This allows you to store the data in the most appropriate form, knowing that you can present it in many different ways with many different formats. The Institute-supplied formats can handle many of the most common representations, but the FORMAT procedure is available to produce formats to meet your customized needs.

SAS is a registered trademark of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.