Paper 143-2010

Using SAS® Formats: So Much More than "M" = "Male"

Pete Lund, Looking Glass Analytics, Olympia, WA

Abstract

Formats in SAS® can be used to change the way that a value is displayed. There are numerous formats supplied by SAS for controlling the way dates, times, numbers, currencies and other types of values appear. You can also use PROC FORMAT to build your own formats and create labels for your data values.

This paper will look at a number of features of PROC FORMAT: using value ranges, having open-ended ranges (with the LOW and HIGH range values), capturing left-overs (with OTHER), nesting one format inside another, controlling the sort order of the formatted values (with the NOTSORTED option), and allowing a value to fall into more than one range (with the MULTILABEL option).

Formats allow much more than just changing the labels that are displayed for your data. SAS procedures analyze data at the formatted value level, which means that you can do different levels of analysis just by changing the format, without having to create new variables. We'll look at different ways to use formats in the DATA step and procedures.

What is a Format?

A format is, most simply put, a method of assigning a label to a value. SAS supplies over 130 formats (in v9.2) for labeling dates, currencies, binary/octal/hexadecimal values and others. For example, if you have a SAS date value of 18309 you could use formats to display it as 02/16/2010 or as February 16, 2010 or as 2010Q1.

You can also create your own formats with PROC FORMAT to label your data as needed. Displaying "M" as "Male" or "CA" as "California" or a value between 1 and 100 as "Low" is easy to do with a format. In either case, using SAS-supplied or user-written formats, all of this is done without changing the underlying data value, just the way that it is displayed.

Why Use a Format?

The obvious answer is that you can give more meaningful labels to data, like "Male" rather than "M" – but there's so much more. You can have more than one format that could be assigned to a variable at different times. A state variable could be formatted to display the state name, the capital city, the SAS region, etc. There's no need for storing multiple variables, just have different formats to display the label needed for the particular need.

Formats can often add some "intelligence" to the data. This can readily be noted in formatting dates and times, which are simply stored as the number of days since January 1, 1960 and the number of seconds since midnight, respectively. As noted earlier, 2/16/2010 is much more meaningful than 18309.

Perhaps the most powerful use of formats is that SAS procedures will do analysis at the formatted value level. So, that means I can do a PROC FREQ on a SAS date variable at the yearly, quarterly, monthly or daily level just by changing the format. Again, there is no need for multiple variables, just multiple formats.

Format Names

Before we look at how formats are used, let's quickly go over format names. The name of a format consists of three parts; the "name", width and decimal portion. For example, the components of three of the SAS-supplied formats are:

Example usage	Format name (required)	Width (required)	Decimal portion (optional)	Original value	Formatted Value
DOLLAR9.2	DOLLAR	9	2	1234.56	$1,234.56
MMDDYY10.	MMDDYY	10		18309	2/16/1996
$UPCASE.	$UPCASE	(default is 8)		a test	A TEST

There are a few things to notice about the format names.

- There is a leading $ for formats that act on character variables.
- There is always a ., even if no decimal portion is required. Obviously, dates and character strings will not have a decimal portion to the display, but the period is always part of the format specification.
- There's always a width, even if you don't specify it. This determines the width of the displayed output and every format has a default width. There is also a specified minimum and maximum width.

In the examples above, both the $DOLLAR and the MMDDYY formats place delimiters and other characters in the displayed label. When using SAS-supplied formats, it's important to keep in mind that any "special" characters that the format places in the value are part of the width. You can look at the SAS documentation to see what the effect of different width specifications will be. For example, here's what the output of the MMDDYY format is with different widths specified (all formatting 18309):

MMDDYY10.	02/16/1996	10-character output
MMDDYY9.	02/16/96	No room for 4-digit year (same result with mmddyy8.)
MMDDYY7.	021696	No room for slashes (same result for mmddyy6.)
MMDDYY5.	02/96	Just the month and year
MMDDYY4.	0296	Just the month and year, but no room for the slash

,,,and so on. You can see that it pays to look at the documentation. Also, note that using MMDDYY1. would have resulted in an error, as the minimum width defined for the MMDDYY format is 2.

For the most part, changing the width of formats that are created with PROC FORMAT will simply result in truncation of the label.

Using a Format – the FORMAT Statement

The FORMAT statement, in either a procedure or a DATA step, associates a format with a variable. If it's used in a datastep, the format is permanently associated with the variable and will be used whenever values for the variable are displayed.

```
data TestFormat;
  format dob MMDDYY10.;
  dob = 18309;
  put dob=;
run;

dob=02/16/2010

data _null_;
  set TestFormat;
  put dob=;
run;

dob=02/16/2010
```

In the simple datastep seen here, the format MMDDYY10. is associated with the variable DOB. Notice that the result PUT statement in the log is the formatted result. The real value of the variable is still 18309.

When the dataset is used in the second datastep, note that the log note still shows the formatted value, even though there is no format specification – the format has been permanently assigned to the variable.

It's important to remember that the format only affects how a variable is displayed. The underlying value remains the same and any arithmetic calculations, statistical analysis, comparisons or assignments will use the "real" value

There is a FORMAT statement in most procedures as well. It too assigns a format to a variable, but that association is for the duration of the procedure only. Suppose that we had a dataset containing a date-of-birth variable, called DOB, for which no format had been assigned when the dataset was created. If we run a PROC FREQ on the variable, we will see the raw SAS date values. We can add a FORMAT statement to the PROC FREQ which will not only change the way the values are displayed, but will determine the level of analysis as well. Notice the difference in the three examples below.

No format

```
proc freq data=DOBrecords;
  table dob;
run;
```

```
   DOB     Frequency      Percent
_________________________________

  8156             1         0.02
  8161             1         0.02
  8168             2         0.03
  8181             7         0.11
  8185             2         0.03
  8189             1         0.02
  8194             1         0.02
...plus more
```

MMDDYY format

```
proc freq data=DOBrecords;
  table dob;
  format dob mmddyy10.;
run;
```

```
   DOB              Frequency      Percent
__________________________________________

 05/01/1982                 1         0.02
 05/06/1982                 1         0.02
 05/13/1982                 2         0.03
 05/26/1982                 7         0.11
 05/30/1982                 2         0.03
 06/03/1982                 1         0.02
 06/08/1982                 1         0.02
...plus more
```

YEAR format

```
proc freq data=DOBrecords;
  table dob;
  format dob year.;
run;
```

```
   DOB     Frequency      Percent
_________________________________

 1970            235         3.83
 1971            227         3.70
 1972            167         2.72
 1973            227         3.70
 1974            170         2.77
 1975            191         3.11
 1976            239         3.90
...plus more
```

The output from the first two procedures have the same number of rows, all we've done is changed the way that the date of birth is displayed. But, the third result is different. Simply by changing the format we've changed the level of the analysis. We now have counts by year, without having to create a variable with the year of birth.

Also, remember that a FORMAT statement used in a datastep permanently assigns the format to the variable. If we had assigned MMDDYY10. to the variable in a datastep, the code from column 1 above (no FORMAT) would have produced the output in column 2 (with the formatted results). A FORMAT statement used in a procedure will override a permanently assigned format, for that procedure only.

Multiple variables and formats can be referenced in a single FORMAT statement. A format will be assigned to all the variables that precede it on the statement. The following are equivalent in assigning formats to the three variables:

Multiple FORMAT statements

```
  format DOB year.;
  format BookingDate year.;
  format Bail dollar9.2;
```

Single FORMAT statement

```
  format DOB BookingDate year. Bail dollar9.2;
```

In both cases the YEAR. format is assigned to the DOB and BookingDate variables and DOLLAR9.2 to the Bail variable.

Using a Format – the PUT Statement

It's already been shown that the formatted value of a variable will be displayed to the log with a PUT statement (see PUT DOB= above). In addition, a format can be specified directly on the PUT statement. If more than one variable is listed on the PUT statement, each must have an explicit format reference.

Using a PUT statement with formatted values is very handing for debugging programs. Also, keep in mind that the same variable can be referenced more than once on a PUT and each reference can have its own format assignment. This too can be handy for some debugging and logging purposes. The example here shows this.

```
data _null_;
  DOB = 18309;

  put DOB= @12 DOB mmddyy10. @24 DOB year. @30 DOB yyq.;
run;
```

```
DOB=18309  02/16/2010  2010  2010Q1
```

The DOB variable is display four times, all with different formatting. It's important to remember that unlike the FORMAT statement, each variable must have an explicit format reference on the PUT statement. The following log note is probably not what was wanted:

```
data _null_;
  DOB = 18309;  FirstAppt = 18334;

  put DOB= FirstAppt= mmddyy10.;
run;
```

```
DOB=18309 FirstAppt=03/13/2010
```

Notice that only the second variable is formatted. Another reference to MMDDYY10. should probably have been placed after DOB as well.

Using a Format – the PUT Function

It's been noted numerous times already that assigning a format to a variable, either permanently in a datastep or temporarily in a procedure, does not change the underlying value of the data. However, what if we wanted to create a variable that had the formatted value as its "real" value? We can do this in an assignment statement with a PUT function.

```
data DOBtest;
  format DOB mmddyy10.;

  DOB = 18309;

  CopyDOB = DOB;
  FmtDOB = put(DOB,mmddyy10.);

  put DOB= CopyDOB= FmtDOB=;
run;
```

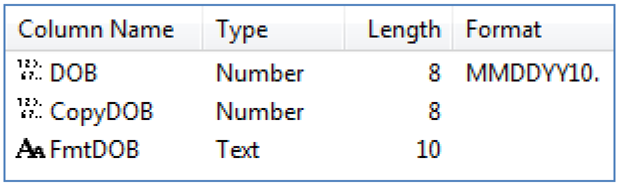
```
DOB=02/16/2010 CopyDOB=18309 FmtDOB=02/16/2010
```

The syntax of the PUT function is simple:
PUT(variable name,format name)

The resulting value is always a character variable and it contains the formatted value of the variable named in the function.

The example here shows this, plus reiterates what has been discussed so far. First, notice that were assigning the MMDDYY10. format to the variable DOB, which has a real value of 18309. Then, we create a new variable, CopyDOB, that has the same value as DOB.

Finally, we use the PUT function to create a new variable, FmtDOB, that contains the formatted value of DOB. When we display the values with the PUT statement at the bottom it may seem that the DOB and FmtDOB values are the same. But, they are not. DOB is a numeric variable with a value of 18309 that is formatted to display as 02/16/1996. The variable FmtDOB is a character variable whose value is the text string "02/16/1996". The variable attributes, shown to the right, confirm this.



Column Name	Type	Length	Format
DOB	Number	8	MMDDYY10.
CopyDOB	Number	8	
FmtDOB	Text	10	

Creating Your Own Formats – PROC FORMAT

PROC FORMAT is used to create formats. The VALUE statement names the format and defines the value-label pairs. It can be as simple as this. Notice that the there is only one semicolon, at the end of the list of values. In this example, the $ at the beginning of the format name indicates that it is character format and is expecting character values as input ("M",'F").

```
proc format;
  value $Gender
    'M' = 'Male'
    'F' = 'Female';
run;
```

This format can be used in any of the ways already demonstrated using SAS-supplied formats. A discussion of how to make user-defined formats available across SAS sessions will come later.

If more than one value is assigned to the same label, there can be either separate entries or a single entry with the values separated by commas. In the following examples, charges that are felonies (F) or probable cause (P) are assigned the label "Felony" and charges that are misdemeanors (M) are assigned the label "Misdemeanor.' The two formats produce the same results. The coding style used is usually a matter of personal style. Aligning the label portion of the format can make it easier to scan for accuracy, but is not required.

Single value per entry

```
proc format;
  value $FelMisd
    'F' = 'Felony'
    'P' = 'Felony'
    'M' = 'Misdemeanor';
run;
```

Multiple values per entry

```
proc format;
  value $FelMisd
    'F','P' = 'Felony'
    'M'     = 'Misdemeanor';
run;
```

It is often the case that multiple values will be assigned the same label. In this case "F" and "P" are assigned the label "Felony." It makes sense, however, that a single value can only appear once on the left side of the assignment. For example, you could not have "F" = "Felony" and "F" = "Other". This would generate an error and the format would not be created. (There is a way to have multiple labels for the same value – that special case will be discussed later in the paper.)

Formats can also be created to assign labels to numeric values. The only differences in the syntax are that there is no dollar sign on the format name and there are no quotes on the values (but the labels are quoted). The rules for multiple values in a single label assignment are the same as they are for character formats – simple separate multiple values with commas. The Grade. format shown here is a simple example of assigning character labels to numeric values.

```
value Grade
  0 = 'F'
  1 = 'D'
  2 = 'C'
  3 = 'B'
  4 = 'A';
```

What's a Hit?

What constitutes a match to a format depends on the type. For character formats it's pretty simple: the data value must match on case, length and justification. For numeric formats it's almost as easy, maybe even easier. The data values must match exactly to the format values, within a given "fuzz factor." The default fuzz is 1E-12, which is satisfactory in most cases.

If a value does not match to any value in the format, the original value is returned, truncated to the length of the longest label. So, if your data had a value of "X" in the ChargeType variable and the $FelMisd. format defined above was used, the result would be "X". For numeric formats, what's displayed on a non-hit is a little trickier to describe.

Again, if there is not a match on the value, the original value is returned. If there are more digits in the original value than there are in the longest label, asterisks are returned. Here are the results of different values and the labels assigned by the Grade. format shown above:

Value	Displayed Label	Note
1	D	Shows the formatted label for this value
5	5	There is no label defined for this value, so the original value is retained
10	*	These is no label, but the original value is longer than the longest label defined, so an asterisk is displayed
1.5	2	There is no label and the rounded integer portion of the original value is returned

There is a way to capture those data values that do not match to any defined values in the format. The special value OTHER (no quotes) defines a catch-all value for all data values that are not in the format.

```
value Grade
   0   = 'F'
   1   = 'D'
   2   = 'C'
   3   = 'B'
   4   = 'A'
 other = 'X';
```

The table above would be quite different if Grade. format looked like this (see left). The values 5, 10 and 1.5 listed in the table above would all be assigned the value ‘X’ rather than some representation of the original value.

It should be noted that missing values are not captured by OTHER. An explicit reference to missing values (‘ ‘ or .) must be made in the format definition to assign a label to a missing value.

Ranges of Values

In addition to uniquely specified values, ranges of values can be used in the value statement. Both character and numeric formats can have ranges of values, though it is much more common with numeric formats. To specify a range, simply separate the upper and lower values with a dash. Ranges cannot overlap, just as a single value cannot be assigned to more than one label.

For example, suppose all items with bail between $1 and $500 are to be labeled “Low” and those between $501 and $1000 are to be labeled “High.” This format would accomplish that.

```
proc format;
  value BailGroup
    1 - 500    = 'Low'
    501 - 1000 = 'High';
run;
```

By default, the range end values are included in the range. With non-integer values, it is often necessary to specify whether the range end(s) should be included. A problem with the format above is with values between 500 and 501, like 500.50. This isn’t defined by any of the ranges and will not be formatted – the original value would be returned. (In this case, the value 500.50 would return 501, since only four characters of the original value are returned. See note above.)

There are a few ways to deal with this. We could change the ranges to be more precise, like 1-500 and 500.01-1000. As noted above, ranges cannot overlap. However, range ends can touch – we could have 1-500 and 500-1000. When ranges to touch, the label associated with the first range is assigned to the touching value. So, 500 would get the label associated with 1-500.

```
proc format;
  value BailGroup
    1 -< 500   = 'Low'
    500 - 1000 = 'High';
run;
```

Both of these methods have their drawbacks. We can resolve this issue more cleanly by using a special range-end exclusion character (<). A < on either side of the dash means to exclude the value on that side of the dash from the range. The above format can be changed, as shown here, now puts everything from 1 up to, but *not including*, 500 into the “Low” group and everything between 500 and 1000 into the “High” group.

The range-end exclusions can be placed on either, or both, sides of the dash to exclude either, or both, of the range-end values.

One final note here – a single format definition can contain any combination of single values, multiple values or ranges of values. Use whatever is easiest to code and maintain.

Special Format Values

There's another issue with the BailGroup. format example above – what if we don't really know what the lowest or highest values might be? We could use ridiculously low and high values to ensure we capture everything, or use the special LOW and HIGH range values.

Replace this…

```
proc format;
  value BailGroup
     -999999-< 500 = 'Low bail'
    500 - 9999999 = 'High bail';
run;
```

…with this

```
proc format;
  value BailGroup
    low -< 500 = 'Low bail'
    500 - high = 'High bail';
run;
```

LOW means the lowest value possible (excluding missing) and HIGH means the highest value possible. It's a much more elegant solution to the problem of data ranges.

Formats Within Formats

Often times it's desirable to just tweak an existing format. The label of a format entry can be a reference to another format. Simply enclose the name of another format, including any necessary width and decimal specification, in square brackets.

For example, in this example, notice that the values between 50,000 and 200,000 use the SAS-supplied DOLLAR12.2 format and values below and above that have a note. The results of this format used in a PROC REPORT (see DEFINE statement) is also seen.

```
proc format;
  value ValidAmt
    low -< 50000    = 'Under $50,000, Watch'
    50000 - 200000    =  [dollar12.2]
    200000 <- high = 'Over $200,000, OK';
run;
```

```
  define sales / mean '' format=ValidAmt.;
```

```
Asia                       Under $50,000, Watch
Canada                      $115,019.24
Middle East                Over $200,000, OK
South America              Under $50,000, Watch
United States               $137,599.65
```

It's very important to remember the square brackets around the embedded format name. If you don't, the name of the format becomes the label – in this case the text "dollar12.2" would the label for all values between 50000 and 200000.

Ordering of Procedure Output

When formatted variables are used in procedures, values can be displayed in either formatted or unformatted order. By default most procedures display values in sorted, unformatted order. The ORDER= procedure option controls the order in which values will be displayed. For most procedures the default is ORDER=INTERNAL – the unformatted order of the raw data.

This is often a non-issue if the unformatted and formatted order are the same. For example, if we format 'M' = 'Male' and 'F' = 'Female', both orders are the same. However, this can often be confusing if the unformatted values are not in the same order as the formatted values. For example, consider this format for offense code values.

```
proc format;
  value $ChargeType
    '23' = 'Property Crimes'
    '35' = 'Drugs'
    '94' = 'DWLS'
   other = 'Other';
run;
```

There are a couple potential order issues here. First, Drugs is the lowest alphabetical formatted value, but not the lowest numerical value. Second, if the "other" catch-all category contains values lower than '23', it will be first in the unformatted list.

In fact, if we use this format in a procedure (PROC FREQ in the following examples) we can see the default order may not be what we want. Just as we suspected, the OTHER category comes out on top because there were values of less than '23' that were assigned to "Other." The remaining values are in unformatted order ('23', '35', '94'). We can use ORDER=FORMATTED on the procedure to change to the formatted order, but this is still not what we want.

Default (ORDER=INTERNAL)

```
ChargeCode           Frequency      Percent
───────────────────────────────────────────
Other                     6299        58.97
Property Crimes           1020         9.55
Drugs                     1488        13.93
DWLS                      1874        17.55
```

With ORDER=FORMATTED

```
ChargeCode           Frequency      Percent
───────────────────────────────────────────
DWLS                      1874        17.55
Drugs                     1488        13.93
Other                     6299        58.97
Property Crimes           1020         9.55
```

What we really want is the order that we coded in the PROC FORMAT, with OTHER as the last category. Until v9, there were two methods for doing this. First, create a new variable that has values that will be in the right unformatted order and then format that value to the label we want. In the above example, we create a new variable based on ChargeCode: '23' = 1, '35' = 2, '94' = 3 and everything else = 4. Then, we format values 1 - 4 according to the above format and our procedures will display in the right order with ORDER=INTERNAL.

The second method is to contrive labels so they sort in the correct order. Adding spaces to the front of the label strings or prefixing labels with a number or letter are ways to get the labels in order for display with ORDER=FORMATTED.

But, now is a better solution that can sometimes be used. The NOTSORTED option on PROC FORMAT specifies that the physical order of the values/ranges should be maintained for display purposes. Unfortunately, only certain procedures can also take advantage of the unsorted order. At this time, PROC MEANS, SUMMARY, REPORT and TABULATE can display formatted values in the NOTSORTED order. The decision to support NOTSORTED is a procedure-level decision. The jury is still out on whether additional procedures will support this option.

Note: There is also an ORDER=DATA option that will order the output according to how the data is physically stored in the dataset. Whatever value is on the first observation will be the first displayed. Whenever a new value is encountered it will be next in the displayed output.

PROC FORMAT Options – NOTSORTED

The NOTSORTED option is a statement level option which is placed on the VALUE statement, simply place the keyword in parentheses following the format name. If the format is used in a procedure the formatted values will display in the order specified in PROC FORMAT. There are also some options that must be set on the reporting procedure to get the desired order.

```
value $ChargeType (notsorted)
  '23' = 'Property Crimes'
  '35' = 'Drugs'
  '94' = 'DWLS'
 other = 'Other';
```

To take advantage of the NOTSORTED option, the PRELOADFMT and ORDER=DATA options must be set in the reporting procedure. The PRELOADFMT option tells SAS to load the formatted values of the variable before the procedure starts its work.

Used in conjunction with PRELOADFMT, ORDER=DATA specifies that the results should be displayed in order defined in the PROC FORMAT step. It would seem that we would want to specify ORDER=FORMATTED, to keep the newly defined NOTSORTED formatted order. However, the correct syntax is ORDER=DATA.

In PROC REPORT, these options are placed on the DEFINE statement for the variable to which they will apply. If they are not, the NOTSORTED option is ignored and the output will be the same as the default.

Without options (same result as before)

```
proc report data=Charges nowd;
  columns ChargeCode N;
  define ChargeCode /
     group format=$ChargeType.;
  define n / format=comma5.;
run;
```

```
Charge NCIC              N
DWLS                 1,874
Drugs                1,488
Other                6,299
Property Crimes      1,020
```

With options (output in desired order)

```
proc report data=Charges nowd;
  columns ChargeCode N;
  define ChargeCode /
    group format=$ChargeType. preloadfmt order=data;
  define n / format=comma5.;
run;
```

```
Charge NCIC              N
Property Crimes      1,020
Drugs                1,488
DWLS                 1,874
Other                6,299
```

The results on the right are now in the desired order, the same order as was coded in the PROC FORMAT.

PROC FORMAT Options – MULTILABEL

In the past (prior to v9), the only way to assign more than one formatted label to a value was to have more than one format. For example, let's suppose that we have a format with county names for our five state county. We also have a format that groups together all but the largest county – those formats and the results of PROC TABULATE using those formats are shown below.

The NOTSORTED option was used on both formats to maintain the desired order. Also, there are a couple things about the summarized format (MyCounty) that might seem odd. First, the Washington County label has a leading space, when it might seem like just the name would do. Secondly, all of the other values were listed for the Rest of State entry when it seems like OTHER= would have been the more obvious choice. There are reasons for both of these that will be discussed later.

We had to run PROC TABULATE (or some other procedure) twice to get a report that contains both the detail and summary information. But, we now have the option of creating both sets of values at the same time using the MULTILABEL option on the VALUE statement.

Detail format

```
  value CountyDetail (notsorted)
    5 = 'Washington County'
    3 = 'Lincoln County'
    2 = 'Jefferson County'
    4 = 'Roosevelt County'
    1 = 'Grant County';
```

	Total	Pct
Washington County	4,341	70.77
Lincoln County	613	9.99
Jefferson County	337	5.49
Roosevelt County	447	7.29
Grant County	396	6.46

Summarized format

```
  value CountySummary (notsorted)
          5 = ' Washington County'
    1,2,3,4 = 'Rest of State';
```

	Total	Pct
Washington County	4,341	70.77
Rest of State	1,794	29.23

```
   value CountyBoth (multilabel notsorted)
               5 = 'Washington County'
               3 = 'Lincoln County'
               2 = 'Jefferson County'
               4 = 'Roosevelt County'
               1 = 'Grant County'
               5 = ' Washington County'
         1,2,3,4 = 'Rest of State';
```

Notice now that we have only one format, CountyBoth, but the range values are repeated. For example, the value of 1 (one) goes into both “Grant County” and “Rest of State”. Obviously this breaks the rules that we’ve discussed so far about values mapping to more than one label. Well, the MULTILABEL option allows us to do just that. Unfortunately, not all procedures allow multi-label formats to be used. At this point, only the TABULATE, MEANS and SUMMARY procedure support multi-label formats.

Prior to version 9.1, the MULTILABEL and NOTSORTED options did not work together. Beginning with 9.1 the two options do work together and provide a great deal of power is displaying the output from multiple label formats. Both the options go in the parenthesis on the VALUE statement.

The MLF option is used in the reporting procedure to allow for processing of formats with multiple labels. When we want to use this format in a procedure we need to specify the appropriate options for both the MULTILABEL and NOTSORTED options. The PROC TABULATE code, with the needed options, and the resulting output are shown below.

PROC TABULATE code

```
proc tabulate data=Bookings;
  class County / preloadfmt order=data mlf;
  table County='',all=''*
     (N='Total'*f=comma5. colpctn='Pct');
  format County CountyBoth.;
run;
```

So, with a single format and a single run of the procedure the analysis is done at two levels. Note that the percentages add up to 100 in each “half” of the output and all the counts in the table add to the appropriate values.

Output

	Total	Pct
Washington County	4,341	70.77
Lincoln County	613	9.99
Jefferson County	337	5.49
Roosevelt County	447	7.29
Grant County	396	6.46
Washington County	4,341	70.77
Rest of State	1,793	29.23

Notice that all of the format-specific options go on the CLASS statement for the variable to be formatted. PROC TABULATE allows for multiple CLASS statements if different format options are needed for different variables.

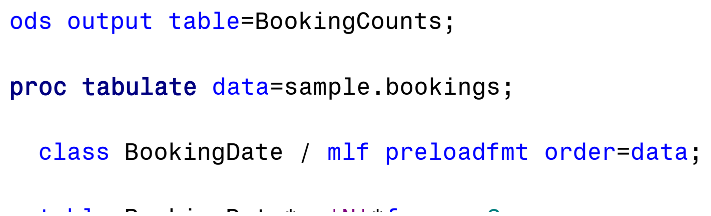
Remember the little quirks in the summary portion of the format (leading space and not using OTHER)? The reason we needed to do this is because there is no way to distinguish the “groups” of entries that make up the two parts of our format. The building of a format is based on creating groups of values that go with unique labels. So, if we had 1 = “Washington County” in the format twice (once for detail and once for summary), those two entries would be treated like any other case where the same label is associated with “multiple” values. The resulting output would just have one row for “Washington County.” Adding the leading space to the name makes the two labels different – fortunately, in most output destinations the leading space is ignored.

What out OTHER? Well, the OTHER entry is going to look at the rest of the format as a whole and find values that did not match to any other entry. In this case, all the values did match to the individual entries in the “detail” portion of the format and OTHER would have no hits. By specifying all the values we make sure that the multilabel specification works.

Using MULTILABEL to Prepare Data

Maybe even more useful than for display, multi-label formats can be used in procedures to create datasets to be used in further processing. Let’s say we had a job that needed to create a dataset with month-to-date, quarter-to-date and year-to-date counts of bookings on September 19, 2009. We can use a multilabel format, specifying each of the date ranges, to do this in one pass through PROC TABULATE.

```
value MultiDates (notsorted multilabel)
  '1jan2009'd - '19sep2009'd = 'YTD'
  '1jul2009'd - '19sep2009'd = 'QTD'
  '1sep2009'd - '19sep2009'd = 'MTD';
```

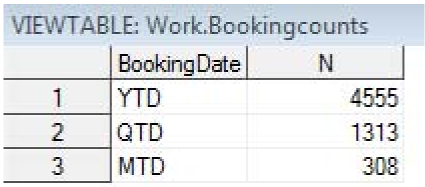


```
ods output table=BookingCounts;

proc tabulate data=sample.bookings;

  class BookingDate / mlf preloadfmt order=data;

  table BookingDate*'N'*f=comma8
```



VIEWTABLE: Work.Bookingcounts

	BookingDate	N
1	YTD	4555
2	QTD	1313
3	MTD	308

Note that the variable name (BookingDate) in the output dateset is the same as in the input dataset, even though it has changed from a numeric (containing a SAS date) to a character variable (containing the format label). Also, note that since PROC TABULATE does not have an output option, ODS OUTPUT was used to create the output dataset.

The resulting dataset could then be used in further processing or reporting. In the real world we would probably be running this job daily and would not want to hard-code the date ranges in the format. We can take advantage of the fact that PROC FORMAT can use macro variable references in ranges, values and labels. So, our daily job has a macro variable called &Today, which contains the current SAS date value. We can rewrite the format as follows to contain the correct date ranges for everyday.

```
value MultiDates (multilabel notsorted)
  %sysfunc(intnx(year,&Today,0)) - &Today     = 'YTD'
  %sysfunc(intnx(quarter,&Today,0)) - &Today = 'QTD'
  %sysfunc(intnx(month,&Today,0))- &Today    = 'MTD';
```

The INTNX function returns a SAS date at some given interval from the base date passed to it. In this case, all the rows of the format use INTNX based on the value of &Today. In the YTD line, we ask for the first day of the year 0 years before today (that would be the first of the year in which &Today falls). In the QTD line we ask for the first day of the quarter and the MTD line asks for the first day of the month. So, no matter which day the job runs, the format will have the correct date ranges and our dataset will contain the correct values.

Conclusion

I hope you can see that formats can be used for a lot more than just assigning a label to a value. The ability to aggregate on formatted values and control the order of report output is very powerful.

PROC FORMAT can do a lot more as well. I invite you to take a look at the SAS documentation and other papers follows to learn more.

Online SAS Documentation , *PROC FORMAT*,
http://support.sas.com/documentation/cdl/en/proc/61895/HTML/default/a000063536.htm

SUGI/SGF paper search: http://lexjansen.com/sugi/ has the most comprehensive collection of papers (complete papers back to 1995 and even a few conferences before that).

Author Contact Information

Feel free to make suggestions or ask questions!

Pete Lund
Looking Glass Analytics
215 Legion Way SW
Olympia, WA 98501
pete.lund@lgan.com
(360) 528-8970

Acknowledgements

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are registered trademarks or trademarks of their respective companies.