Paper 174-2008

I Can Do That With PROC FORMAT

Jonas V. Bilenas, JP Morgan Chase, Wilmington, DE

ABSTRACT

In this tutorial we illustrate how to create your own user-defined FORMATS, INFORMATS, and PICTURE FORMAS. We describe the many applications of user-defined FORMATS including extracting data, assigning values to variables, reporting, table look-ups, and extracting data based on keys without sorting data. We also point out the quirks of the FORMAT procedure, such as why you need to watch your fuzz factor, how to generate FORMATS from CNTLIN data sets, watching out when using the w.d INFORMATS on formatted variables without decimal points, and how to use the MULT and ROUND options in creating PICTURE FORMATS. After this tutorial, you too will say, "I can do that with PROC FORMAT."

INTRODUCTION TO INFORMATS AND FORMATS

Informats are used to read data into SAS and formats are used to write out data to files and/or reports. Let's take a look at some examples. We can specify INFORMATS in an INPUT statement to read data of many INFORMATS (character, numeric, and date). Here is an example reading data using INFORMATS:

```
options nocenter;

data test;
  input @1 id $char4.
        @5 age 2.
        ;
  datalines;
A14X54
B16Y24
C24Q19
A33A66
;
run;

proc print data=test;
run;
```

Output from the PROC PRINT:

```
Obs     id     age

 1     A14X     54
 2     B16Y     24
 3     C24Q     19
 4     A33A     66
```

We can look at an example of using FORMATS in the PROC PRINT. Let's change the PROC PRINT portion of the above code as follows:

```
proc print data=test;
  format _numeric_ z6.2;
run;
```

OUTPUT:

```
      Obs     id        age

       1     A14X    054.00
       2     B16Y    024.00
       3     C24Q    019.00
       4     A33A    066.00
```

Note that the Z*w.d* FORMAT is used to print leading zeros. This can be useful for zip code data if stored as numeric. Some general rules:

1. Character FORMATS and INFORMATS start with a $.
2. All FORMATS and INFORMATS must contain a . to differentiate if from a variable name.
3. w in a w.d format specification counts the number of bytes, including the decimal point.
4. d in a w.d format specification count the number of digits after the decimal point.
5. If the digit contains a decimal point, you don't need the d specified.

Example of item 5:

```
options nocenter;

      data test;
        input @1 id $char4.
              @5 age 2.
              @7 weight 5.
              ;
        datalines;
      A14X54132.4
      B16Y24278.1
      C24Q19 95.6
      A33A66165.3
      ;
      run;

      proc print data=test;
        format _numeric_ z6.1;
      run;

OUTPUT:

      Obs     id        age    weight

       1     A14X    0054.0    0132.4
       2     B16Y    0024.0    0278.1
       3     C24Q    0019.0    0095.6
       4     A33A    0066.0    0165.3
```

WHY CREATE USER DEFINED FORMATS: TABLE LOOK-UPS

Building a user defined FORMAT can be viewed as a table lookup where VALUES are mapped to LABELS. Let us look at some table lookup concepts.

Typical lookup tables use 1-to-1 or many-to-1 mappings. As an example of a 1-to-1 table lookup, we have a data set that has a variable that codes up credit approval/decline decision codes. When we generate reports of approvals and declines, we wish to map decision code values into literal labels. The 1-to-1 mapping is illustrated here:

'a' = 'Approve'
'd' = 'Decline'

If we have many approval codes and many decline codes, these can be mapped or assigned or grouped to the appropriate label in a many-to-1 mapping. For example:

'a1', 'a2', 'a4' = 'Approve'
'd1', 'd6' = 'Decline'

In the next section, we will look at how we might generate table lookups in SAS. The first method relies on the DATA step and the second approach will build the lookup using the more efficient method with PROC FORMAT.

TABLE LOOKUP USING A DATA STEP

For this example, we will look at groupings of credit bureau risk scores. These scores are often used in credit decisions. One such score has integer values from 370 to 870 with exception scores outside this range. Higher values of the score relates to better credit quality and reduced risk.

We wish to run a frequency distribution on individuals with scores grouped into 3 classes; 370-670, 671-870, and un-scored. This is an example of a many-to-1 mapping where we will need to group scores into 3 categories. A beginning programmer would often handle this by creating another SAS data set where a new variable is generated to assign membership into categories. This new data is then used in a PROC FREQ to generate the report. Here is some code generated by the SAS programmer:

```
data stuff;
  set cb;
  if 370<= score <= 670 then group='670-';
  else if 670 < score <= 870 then group='671+';
  else group='unscored';
proc freq data=stuff;
  tables group;
  run;
```

Results from the code are:

GROUP	Frequency	Percent	Cumulative Frequency	Cumulative Percent
671+	623	10.7	623	10.7
670-	5170	89.2	5793	99.9
unsc	5	0.1	5798	100.0

The code did the job, but it required that a new DATASTEP be created and the label 'unscored' was truncated to 'unsc'.

TABLE LOOKUP USING PROC FORMAT

Using a user defined FORMAT saves some processing time and resources. The same problem solved using PROC FORMAT is illustrated here.

```
proc format;
  value score 370 - 670 = '670-'
              670<- 870 = '671+'
              other     = 'unscored'
  ;
proc freq data=cb;
  tables score;
  format score score.;
run;
```

Code returns this output:

```
                                        Cumulative  Cumulative
  SCORE    Frequency    Percent    Frequency     Percent
-------------------------------------------------------------
670-            5170       89.2         5170        89.2
671+             623       10.7         5793        99.9
unscored           5        0.1         5798       100.0
```

Some observations to make:

1. Assignment of the FORMAT occurs in PROC FREQ with a FORMAT statement.
2. I end the format definition with the ';' on a new line. This is just my preference but I find it easier to debug PROC FORMAT code especially if I add more value mappings to the FORMAT later on.
3. The 'unscored' label now appears without truncation.
4. FORMAT name does not have to match the name of the variable you will apply the FORMAT to.

Syntax rules for PROC FORMAT will be reviewed in a later section. Let's look at another application of PROC FORMAT to find unexpected values.

USING PROC FORMAT TO FIND UNEXPECTED VALUES

User defined formats can be used to list out unexpected values. If a range of values are not mapped in a PROC FORMAT, the original values will be returned as labels. Here is an example:

```
proc format;
  value looky 370-870 = '370-870'
  ;
proc freq data=cb;
  tables score;
  format score looky.;
run;
```

Output is as follows:

```
                                Cumulative  Cumulative
  SCORE    Frequency    Percent    Frequency     Percent
-------------------------------------------------------------
370-870        30320       96.0        30320        96.0
   9003         1264        4.0        31584       100.0
```

With the above example we run the risk of truncating the output of values if the values have a width larger than the width of the FORMAT label. For this reason, it is better to use an embedded FORMAT as follows:

```
proc format;
  value looky 370-870 = '370-870'
              other   = [best.]                    ❶
  ;
```

With this code, we embedded a FORMAT within a FORMAT in line ❶. These have to be within brackets or pipes (|).

GENERATING NEW VARIABLES WITH PROC FORMAT

New variables can be assigned within a data step using user defined FORMATS. A nice feature of using FORMATS for generating new variables is that the method can replace IF/THEN/ELSE code. By default, PROC FORMAT will not allow 1-to-many or many-to-many mapping. There are no checks for accidental 1-to-many or many-to-many mapping in IF/THEN/ELSE code within a data step.

In this example we wish to assign a credit line based on a risk score range.

```
proc format;
  value stx  low - < 160 = '1000'
             160 - < 180 = '2500'
             180 - < 200 = '5000'
             200 - < 220 = '7500'
             220 -  high = '9500'
  ;
data scores;
  do score = 10, 160, 170, 180, 200, 210, 220, 230;
    line = put(score,stx.);
    put _all_;
  end;
run;
```

OUTPUT:

```
score=10  line='1000' _ERROR_=0 _N_=1
score=160 line='2500' _ERROR_=0 _N_=1
score=170 line='2500' _ERROR_=0 _N_=1
score=180 line='5000' _ERROR_=0 _N_=1
score=200 line='7500' _ERROR_=0 _N_=1
score=210 line='7500' _ERROR_=0 _N_=1
score=220 line='9500' _ERROR_=0 _N_=1
score=230 line='9500' _ERROR_=0 _N_=1
```

Note that the LINE variable is saved as character with the code. Using a PUT function will always return a character format. We need to change the LINE assignment as

```
line = input(put(score,stx.),best.);
```

Output:

```
score=160 line=2500 _ERROR_=0 _N_=1
score=170 line=2500 _ERROR_=0 _N_=1
score=180 line=5000 _ERROR_=0 _N_=1
```

WHAT ABOUT A 2 DIMENSION TABLE LOOKUP?

Taking the last example, what if we wanted to offer different lines as a function of score for 20% of the records? Here is sample code:

```
proc format;
  value use  low   -  0.8 = 'stx'
             0.8 < - high = 'sty'
  ;
  value stx  low - < 160 = '1000'
             160 -   179 = '2500'
             180 -   199 = '5000'
             200 -   219 = '7500'
             220 -  high = '9500'
  ;
  value sty  low - < 160 = '1500'
             160 -   179 = '3200'
             180 -   199 = '6500'
             200 -   219 = '8000'
             220 -  high = '10000'
  ; run;
```

```
data scores;
  set bbu.scores;
  fmtuse = put(ranuni(83),use.);                    ❶
  line = input(putn(score,fmtuse),best12.);         ❷
run;
```

Some comments about the code:

❶ I take a uniform random number and assign 'stx' 80% of the time and 'sty' 20% of the time as values to variable FMTUSE using a PUT function.

❷ I use the PUTN function which assigns, as the second argument, the variable containing the FORMAT name you wish to assign. This will dynamically assign the format based upon values of the FMTUSE variable. Note that the second argument in the PUTN function does not end in a dot since the second argument is not a FORMAT but a variable. For character FORMATS use the PUTC function.

More examples of 2-dimensional and 3-dimensional table lookups using PROC FORMAT can be found in SUGI31 paper by Perry Watts, "Using Database Principles to Optimize SAS® Format Construction from Tabular Data".

WATCH THOSE DECIMAL POINTS

As we have seen in the section on using INFORMATS, be careful when you use the decimal indicator. Here is an example where we get unexpected results:

```
proc format;
  value $test 'a' = '1.2345'
              'b' = '1.5432'
              'c' = '100000';
run;

data _null_;
  do i = 'a', 'b', 'c';
    ta = input(put(i,$test.),6.4);
    put _all_;
  end;
run;
```

Output:

```
i=a ta=1.2345 _ERROR_=0 _N_=1
i=b ta=1.5432 _ERROR_=0 _N_=1
i=c ta=10 _ERROR_=0 _N_=1
```

We notice that the last entry mapped to 10 as opposed to 100000. We can rectify this by using the best INFORMAT:

```
ta = input(put(i,$test.),best.);
```

Using a 6 INFORMAT will work as well.

```
ta = input(put(i,$test.),6.);
```

WATCH YOUR FUZZ

When working with small numeric values sometimes results are not as expected. Check the following code and output:

```
proc format;
  value stx  0 - 1e-20  = '1'
             1e-20< - 2e-20  = '2'
  ;
run;
```

```
data scores;
  do score = 0, 2e-20;
    looky = put(score,stx.);
    put _all_;
  end;
run;
```

OUTPUT:

```
score=0 looky=1 _ERROR_=0 _N_=1
score=2E-20 looky=1 _ERROR_=0 _N_=1
```

Note that both values map to LOOKY=1. There is a default fuzz value in PROC FORMAT of 1e-12. To correct the above code, modify the PROC FORMAT section as follows:

```
proc format;
  value stx (fuzz=0)  0 - 1e-20  = '1'
            1e-20< - 2e-20  = '2'
  ;
run;
```

Resulting Output:

```
score=0 looky=1 _ERROR_=0 _N_=1
score=2E-20 looky=2 _ERROR_=0 _N_=1
```

USING PROC FORMAT TO EXTRACT DATA

User defined formats can be used to extract a subset of data from a larger DATASET. Here is an example.

```
proc format;
  value $key '06980'                    = 'Mail1'
             '06990', '0699F', '0699H' = 'Mail2'
             other                      = 'NG'
  ;
data stuff;
  set large.stuff;
  where put(seqnum,$key.) ne 'NG';
```

Note that for this example we are generating a character FORMAT that will map character values. Character FORMATS must start with a '$'.

Let's review some syntax rules for setting up user defined FORMATS in the next sections.

SPECIFYING RANGES OF VALUES IN PROC FORMAT

Ranges of values can be specified in a number of ways and special keywords can be used in the expression of the range.

1. VALUES can be single values or values separated by commas:
   - 'x'
   - 'a', 'b', 'c'
   - 1, 22, 43

2. Ranges (numeric or character) can include intervals such as:
   - A – B. Interval includes both endpoints.
   - A <- B. Interval includes values larger than A through B.
   - A - < B. Interval includes values from A to less than B.
   - A <- < B. Interval does not include either endpoint.

3. Ranges can be specified with special keywords:
   - LOW, HIGH, OTHER, .,' '

4. The LOW keyword does not format missing values for numeric formats. For character formats, LOW includes missing values.

5. The OTHER keyword does include missing values unless accounted for with specification of missing values.

OTHER FORMAT REQUIREMENTS

- For SAS8 and earlier, format names must be 8 characters or less. For SAS9, the number of characters a format name can have is 32. These lengths include the dollar sign required for character formats.
- FORMAT names cannot be identical to existing internal SAS format names.
- FORMAT names cannot end or begin with a number.
- Character FORMATS must begin with a "$".
- INFORMATS can be created in PROC FORMAT with the INVALUE statement. This paper did not touch on the steps to create user informats. To review the INVALUE statement of PROC FORMAT, refer to SAS documentation and/or SAS PRESS book "The Power of PROC FORMAT" (2005, Bilenas).

USING PROC FORMAT FOR DATA MERGES

PROC FORMAT also offers a method of merging large data sets (up to a few million, depending on memory resources) to very large (millions and millions) unsorted SAS data sets or flat files. The method first builds a user defined format from a special data set. The requirements for this data set are that it must not have any duplicates in key fields and have at least these variables:

- FMTNAME: name of format to create.
- TYPE: 'C' for character or 'N' for numeric.
- START: the value you want to format into a label. If you are specifying a range, START specifies the lower end of the range and END specifies the upper end.
- LABEL: the label you wish to generate.

Once the data is generated, a FORMAT is generated from the data and then applied to match records from the larger unsorted SAS data set or flat file. Here is an example of code applied to a large unsorted SAS data set.

```
proc sort data=small out=temp nodupkey force;          ❶
  by seqnum;

data fmt (rename=(seqnum=start));                      ❷
  retain fmtname 'key'                                 ❸
         type  'C'
         label 'Y';
  set temp end=eof;
  output;
  if eof then do;                                      ❹
    start =' ';
    label = 'N';
    HLO   = 'O';                                       ❺
    output;
  End;

proc format cntlin=fmt; run;                           ❻
```

```
data match;                                                    ❼
  set bigfile;
  where put(seqnum,$key.)= 'Y';
run;
```

Some observations on above code:

- The sort of the small DATASET (❶) was done to ensure no duplicates of the key variable, SEQNUM.
- In line ❷ we create the dataset that will be used by PROC FORMAT to generate the FORMAT.
- Also, on line ❷, we need to rename the key variable SEQNUM to START.
- Since FMTNAME, TYPE and LABEL will not change for each record we can use the RETAIN statement starting on line ❸ (RETAIN is more efficient than variable assignment statements).
- Note that we set FMTNAME to 'key'. In this example, the format type is character since the key field on which to match is character. We also assign a value of 'C' to TYPE to indicate that we are setting up a character format. This code will work if we call FMTNAME 'key' or '$key'. Another interesting quirk of PROC FORMAT.
- Beginning with line ❹ we start a DO loop to handle specification of an 'OTHER' condition. The HLO variable specified in line❺ will handle the OTHER specification by setting HLO='O'. It is a good idea to set the START variable to missing to avoid a one-to-many mapping. At the end of the loop, we output another record where LABEL is set to 'N'.
- In ❻, we specify the CNTLIN= option on the PROC FORMAT statement. This will read in the data FMT and generate the FORMAT $KEY that we will then use in the merge-extract DATA step starting on line ❼.

PICTURE FORMATS

PICTURE FORMATS provide a template for printing numbers. The template can specify how numbers are displayed and provide a method to deal with:

- Leading zeros.
- Decimal and comma placement.
- Embedding characters within numbers.
- Prefixes.
- Truncation or rounding of numbers.

Many examples are included in (2005, Bilenas). One example of using PICTURE FORMATS is to add a trailing '%' in PROC TABULATE output when a PCTSUM or PCTN calculation is specified. For this example, we also use the ROUND option so that the number is rounded rather than truncated. This code will print a leading 0 if the percentage is less than 1 (i.e., .67% displays as 0.67%) since we include a digit selector with a value other then 0 to the left of the decimal point. With the two '9' values after the decimal, 2 digits will be displayed after the decimal even if one or more are 0.

```
proc format;
  picture p8r (round) 0-100 = '0009.99%'
  ;
```

The above example will remove negative signs for negative values when applying the format. This may not be an issue in PROC TABULATE when using PCTSUM or PCTN statistics, but you may want to be safe and modify the code as follows:

```
proc format;
  picture p8r (round)
    low - < 0 = '0009.99%' (prefix='-')
    0 - high  = '0009.99%'
  ;
```

CREATING MULTI-LABEL FORMATS

With the introduction of SAS8, we have the capability to map values to more than one label. Multi-label formats can be used in PROC SUMMARY and PROC TABULATE. Let us take a look at an example that we introduced earlier where we are mapping credit decision codes into labels:

'a1', 'a2', 'a4' = 'Approve'
'd1', 'd6' = 'Decline'

We wish to generate a frequency report for each decision and get totals for each category. Here is the code that generates the formats, the hypothetical data, and the summary report.

```
proc format;
  value key low -    0.20 = 'a1'        ❶
            0.20 < - 0.25 = 'a2'
             0.25 < - 0.35 = 'a4'
            0.35 < - 0.80 = 'd1'
            0.80 < - high = 'd6'
  ;
  picture p8r (round)                    ❷
    low - < 0 = '0009.99%' (prefix='-')
    0 - high  = '0009.99%'
  ;
  value $deccode (multilabel notsorted) ❸
         'a0' - 'a9' = 'APPROVE TOTALS'
         'a1'        = ' a1: Approval'
         'a2'        = ' a2: Weak Approval'
         'a4'        = ' a4: Approved Alternate Product'
         'd0' - 'd9' = 'DECLINE TOTALS'
         'd1'        = ' d1: Decline for Credit'
         'd6'        = ' d6: Decline Other'
   ;
run;

data decision;
  do id = 1 to 1000;
    decision = put(ranuni(7),key.);     ❹
    output;
  end;

proc tabulate data=decision noseps formchar='          ';
  class decision/mlf preloadfmt order=data;      ❺
  format decision $deccode.;
  table (decision all)
        ,n*f=comma5.
        pctn='%'*f=p8r.
        /rts=33 row=float misstext=' ';
run;
```

In the PROC FORMAT section of the code we create 3 formats. In line ❶ we generate the format that will assign decision code to records in a SAS data set based on a uniform random number generated in a DO loop (❹).
The second format in line ❷ generates the PICTURE format for displaying percent signs in PROC TABULATE.

The final format in line ❸ is used to generate the multi-label format. Some comments on this format:

1. Note that we must specify the (multilabel) option when generating the format.
2. We can preserve the order of formatted values on tabulate output by specifying the NOTSORTED option in the generation of the FORMAT and PRELOADFMT ORDER=DATA options in line ❺; the CLASS specification in TABULATE.
3. Note that we have labels for each of the 5 decision codes. We also map all codes beginning with the letter 'a' into 'APPROVE TOTALS' and all those beginning with the letter 'd' into 'DECLINE TOTALS'
4. The CLASS statement in TABULATE or SUMMARY (❺) must include the MLF option to generate the multi-label output.

Output from TABULATE:

	N	%
decision		
APPROVE TOTALS	314	31.40%
a1: Approval	163	16.30%
a2: Weak Approval	45	4.50%
a4: Approved Alternate Product	106	10.60%
DECLINE TOTALS	686	68.60%
d1: Decline for Credit	453	45.30%
d6: Decline Other	233	23.30%
All	1,000	100.00%

SAVING FORMATS

Sometimes you may want to save your formats to use in other code or to be used by other users. Use the LIBRARY= option:

```
libname libref ..
proc format library = libref;
  value ..
  ;
```

To use the saved format in a subsequent program without having to enter the FORMAT code, specify a LIBNAME of LIBRARY so that SAS will look for user formats in the LIBRARY.

```
libname library ..
```

You can edit the format by first converting it into a SAS data set using a CNTLOUT option in PROC FORMAT and then edit the data set with PROC FSEDIT. When the changes are made, convert back to a user FORMAT with the CTLIN= option. Here is an example of using the CNTLOUT oprion:

```
proc format library=libref cntlout=SAS dataset;
  select entry;
```

To print out a saved FORMAT library, use the FMTLIB option as indicated:

```
libname library  …
proc format library=library fmtlib;
run;
```

CONCLUSION

The FORMAT procedure allows users to create their own formats that allow for a convenient table look up in SAS. Using PROC FORMAT will make your code more efficient and your output look more professional.

REFERENCES:

- Bilenas, J. "The Power of PROC FORMAT", SAS Press, 2005.
- Watts, P. Using Database Principles to Optimize SAS® Format Construction from Tabular Data, SUGI31, 2006

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Jonas V. Bilenas
JP Morgan Chase Bank
Wilmington, DE 19801
Email: Jonas.Bilenas@chase.com
jonas@jonasbilenas.com

This work is an independent effort and does not necessarily represent the practices followed at JP Morgan Chase Bank.