Paper 001-30

PROC FORMAT – Not Just Another Pretty Face

Lois Levin, Independent Consultant, Bethesda, MD

ABSTRACT

PROC FORMAT, because of its name, is most often used to change the appearance of data for presentation. But PROC FORMAT can be used for much more than the cosmetic transformation of data. It can be a tool for redefining data, recoding data, transforming data for tabulation and reporting, subsetting large data files and more. This paper will present a few applications of PROC FORMAT that go beyond the basic display of data and show that PROC FORMAT has real character that goes deeper than just its pretty face.

INTRODUCTION

There is a wealth of stored formats in SAS® that may be used to express data in many ways. PROC FORMAT provides a method of going beyond those supplied methods to transform data in any way you choose. This paper is not meant to be a tutorial on the use of PROC FORMAT with all of its options and keywords. Rather, it is a collection of interesting applications of PROC FORMAT that will allow you to control the expression and management of data.

The examples included will show how PROC FORMAT can be used to:

1. Group/recode data
2. Identify missing data
3. Display the same variable several ways
4. Perform a table lookup
5. Display negative percent values with negative signs and % signs.
6. Display datetimes
7. Code styles conditionally
8. Extract records from a very large dataset

<u>1 – GROUP/RECODE DATA</u>

PROC FORMAT can be used to define groups of data. If your data are numeric, the groups allow you to report the data as meaningful distributions. The groups also allow data to be recoded based on the group definitions. Groups can be defined and then codes assigned to each group. This could also be accomplished with a sequence of IF statements, but the format will probably take less code and can often run faster. The transformation using one PUT statement is much faster than many IF statements.

In the following example we want to report temperature data by groups of “Low”, “Medium”, “High”. We will use 2 formats -- one to recode the data and one to assign the names. Why not just recode the temperature data as “Low”, “Medium”, or “High”? If we had a large, permanent dataset, we would have to store all of those literal descriptions for every observation. It is much more efficient to recode the data to short character values (Short character values take up less space than standard numeric values), and then use the literals for reporting only. The other reason for using two formats is that the formats will be sorted alphabetically. So our report would show the data as “High”, “Low”, and “Medium” in that order. By coding them in sequence first, we can get them in the order we like.

Example 1 – Program

```
proc format;
     value tempfmt   low -< 61    = ' 1'
                     61  -< 63    = ' 2'
                     63   - high = ' 3'
                     other        = '  '
                     ;

     value $ codefmt
                    '1'='Low    '
                    '2'='Medium'
                    '3'='High   '
                    ;
run;

data pme;
     set pme;

     *---   replace list of IF statements     ---*;
     *        if avgtemp < 61 then code=' 1';
     *   else if avgtemp < 63 then code=' 2';
     *   else if avgtemp <999 then code=' 3';
     *   else code='  ';
     *-------------------------------------------*;

     tempcode=put(avgtemp, tempfmt.);
run;

proc freq data=pme;
     tables tempcode/missing;
     format tempcode $codefmt.;
run;
```

Example 1 – Output

```
                 Example 1 - Group/Recode Data
        Temperatures in Portland, ME - September 2003


                                          Cumulative    Cumulative
 tempcode    Frequency     Percent     Frequency       Percent
 -------------------------------------------------------------
 Low               11       36.67             11         36.67
 Medium             6       20.00             17         56.67
 High              13       43.33             30        100.00
```

2 – IDENTIFY MISSING DATA

A specific case of using formats to group data is this one which divides the data into just 2 groups, missing and non-missing. It is a convenient way to take a quick initial look at your data. This example defines 2 formats – one numeric and one character because we have to distinguish between the numeric missing (.) and the character missing ( ). Then they can be applied to all of the variables in a dataset.

Example 2 - Program

```
proc format;
     value $ missfmt ' '="Missing"
                   other="Not Missing"
     ;
     value  nmissfmt  . ="Missing"
                   other="Not Missing"
     ;
run;

proc freq data=home.ex2;
     tables _numeric_ _character_/missing;
     format _numeric_   nmissfmt.
            _character_ $missfmt.;
run;
```

Example 2 – Output

```
                Example 2 - Identify Missing Data

                            Percent 1

                                        Cumulative    Cumulative
   percent1     Frequency      Percent     Frequency       Percent
----------------------------------------------------------------
Missing              160         1.60           160          1.60
Not Missing         9840        98.40         10000        100.00


                            Amount 2

                                        Cumulative    Cumulative
    amount2     Frequency      Percent     Frequency       Percent
----------------------------------------------------------------
Not Missing        10000       100.00         10000        100.00


                             Date 1

                                          Cumulative    Cumulative
      date1     Frequency      Percent     Frequency       Percent
----------------------------------------------------------------
Missing            10000       100.00         10000        100.00
```

3 – DISPLAY THE SAME VARIABLE SEVERAL WAYS

Here is a case where we have one date variable but we want to display the data in several ways. The format ex3fmt specifies the date as a 4-digit year, a month/year, and also a specific day like 01JAN2003.

This example allows us to see older dates by year, recent dates by month/year, and current dates as a specific day. We are using the technique of **nested formats**, meaning that our format references another format in its definition. The referenced format is coded with brackets (or parentheses and vertical bars, e.g. (|year4.|)) to indicate that it is defined elsewhere. In this case it is just a standard SAS format but it could reference another user-defined format.

Note that the ranges of the dates do not overlap. If you wanted to see different displays of the same data, you could use the MULTILABEL option in PROC FORMAT to define multiple formats for a given date range.

Example 3 - Program

```
proc format;
     value ex3fmt   low-'31DEC02'd=[year4.]
                    '01JAN03'd-'31DEC03'd=[monyy7.]
                    '01JAN04'd-high=[date9.];
run;

proc freq data=&home.ex3;
     tables ex3dat;
     format ex3dat ex3fmt.;
run;
```

Example 3 - Output

```
         Example 3 - Display the Same Variable Several Ways

                              ex3dat

                                           Cumulative    Cumulative
    ex3dat     Frequency      Percent       Frequency       Percent
 --------------------------------------------------------------
 2000             20246         7.86          72272         28.07
 2001             14055         5.46          86327         33.53
 2002             23766         9.23         110093         42.76
 JAN2003          58857        22.86         168950         65.62
 FEB2003          20399         7.92         189349         73.55
 MAR2003           9337         3.63         198686         77.17
 APR2003           2724         1.06         201410         78.23
 MAY2003           6492         2.52         207902         80.75
 JUN2003           6760         2.63         214662         83.38
 JUL2003           7974         3.10         222636         86.48
 AUG2003           4325         1.68         226961         88.15
 SEP2003           3416         1.33         230377         89.48
 OCT2003           7820         3.04         238197         92.52
 NOV2003           8016         3.11         246213         95.63
 DEC2003          11097         4.31         257310         99.94
 01JAN2004           20         0.01         257330         99.95
 15JAN2004            4         0.00         257334         99.95
 02FEB2004           14         0.01         257348         99.96
 28FEB2004           26         0.01         257374         99.97
 04MAR2004           80         0.03         257454        100.00
 15MAR2004            3         0.00         257457        100.00
```

4 – PERFORM A TABLE LOOKUP

One very common and useful application of PROC FORMAT is for table lookups. You can have data in one file and some related data in another file and you can combine them based on a common variable. This type of task is often done with a DATA step and a MERGE statement. But if we do it with formats, the PUT statement that defines the new variable will use much less CPU than a MERGE statement.

In this example we have one dataset with SAILNO and NAME and another with race results by SAILNO. We want a report of race results with the SAILNO and NAME. We are going to do it by creating a format using the SAILNO and use it to lookup the NAME when we read in the race data.

There are several ways to define the format. A VALUE statement could be explicitly defined. If the list of values is long, the data can be read in with a DATA step and a format generated from the data using the CNTLIN option. In this example we have a relatively short list of values (for SAILNO and NAME) so we will create the format by inputting the data into a macro variable and then use that macro variable to define the format value. The macro variable NAMELIST is created by using PROC SQL with the SELECT…INTO: construct to place the list of values of the variable SAILNO into the macro variable. Then the format is defined by referring to the macro variable &NAMELIST. The new variable BOATNAME Is created by formatting the variable SAILNO as $NAMEFMT.

The excerpt from the log shows how the macro variable is interpreted to include the VALUE definition. The report shows the new variable BOATNAME which has been created by using the variable SAILNO with the newly defined format NAMEFMT. It performs the equivalent of:

```
data ex4;
    merge sailname saildat1;
    by sailno;
run;
```

Example 4 – Program

```
proc sql noprint;
     select cats(qt,sailno,qt,eq,qt,name,qt)
     into :namelist separated by ' '
     from sailname;
quit;

proc format;
     value $ namefmt &namelist;
run;

data ex4;
     set saildat1;
     Boatname=put(sailno,$namefmt.);
run;

proc print data=ex4 noobs;
     var Position Boatname Sailno Race1 Race2 Race3 Total;
run;
```

Example 4 – Log

```
38          proc format;
39               value $ namefmt &namelist;
SYMBOLGEN:  Macro variable NAMELIST resolves to 'USA 51518'='LE CYGNE' 'USA
            32'='MULLIGAN' 'USA 34'='BLOCKADE RUNNER' '83445'='FREQUENT FLYER'
            '42449'='PROMISES' '39519'='AMERICAN FLYER' '93004'='DAME BLANCHE'
            '93081'='EUROTRASH GIRL' '23798'='PURSUIT' '93165'='GIJIMI KAKHULA'
NOTE: Format $NAMEFMT has been output.
40          run;
```

Example 4 – Output

```
                    Example 4 - Perform a Table Lookup
                     Using Formats Created from Data
                Top 10 Finishers - Annapolis Race Week 2003

 Position    Boatname           Sailno       Race1    Race2    Race3    Total

     1       LE CYGNE           USA 51518     1        6        5         12
     2       MULLIGAN           USA 32        2        4        9         15
     3       BLOCKADE RUNNER    USA 34        12       3        1         16
     4       FREQUENT FLYER     83445         6        10       2         18
     5       PROMISES           42449         7        2        13        22
     6       AMERICAN FLYER     39519         3        8        11        22
     7       DAME BLANCHE       93004         5        15       3         23
     8       EUROTRASH GIRL     93081         8        5        12        25
     9       PURSUIT            23798         17       13       4         34
    10       GIJIMI KAKHULA     93165         18       9        8         35
```

5 – DISPLAY NEGATIVE PERCENT VALUES WITH NEGATIVE SIGNS AND % SIGNS

SAS has many built-in formats that you can use to display data. But a finite list is never enough so PROC FORMAT supplies the **picture format** capability so that you can design your own format. Where the value statement lets you display data as formatted numerics or literals, the picture statement lets you display numeric data in special patterns using special characters. The SSN and phone number pictures are the most obvious examples of picture formats. They use numbers with dashes in specific patterns. Now consider the *percentw.d* format. It displays numerics as percentages with the percent sign (%) and negative values within parentheses. Suppose you want this format but want negative values shown with a negative sign. Here is a picture format that will do that. The program reads in some data and prints it with no format, so that you can see the raw values, with the stored percent format, and with our very own picture format.

To create the format, we first have to convert the value to a percent and that is what the MULT option does. It moves the decimal point over to the place where we want to display it. Then we need to show the negative numbers with the negative sign and the PREFIX option defines this. So we need to define our format with two kinds of values – the negatives and everything else. The picture itself shows the layout of the number including the decimal point and the percent sign. The 9's indicate that we want a value displayed in that position, even if it is zero. The 0's indicate that those leading zeros may be dropped. If we used '000.99', then a value of .59 would be displayed as 59%. We have also used the ROUND option to ensure that the value is rounded, not truncated, to our designated 2 decimal places.

Example 5 - Program

```
proc format;
    picture pctpic (round) low-<0  ='009.99%'  (prefix='-' mult=10000)
                             0-high='009.99%'  (mult=10000)
     ;
run;

data t1;
     input @1  typenum  $char5.
           @9  var1     9.6
           ;
     cards;
Type1   -0.005855
Type2    0.012498
Type3   -0.047033
;
run;
```

```
proc print data=t1 noobs split='*';
     var typenum var1;
     label typenum="Type"*"----"
           var1="Var1"*"--------";
title 'Report without Format';
run;

proc print data=t1 noobs split='*';
     var typenum var1;
     label typenum="Type"*"----"
           var1="Var1"*"--------";
     format var1 percent8.2;
title 'Report with Percent Format';
run;

proc print data=t1 noobs split='*';
     var typenum var1;
     label typenum="Type"*"----"
           var1="Var1"*"--------";
     format var1 pctpic.;
title 'Report with Picture Format';
run;
```

Example 5 - Output

```
Example 5 - Report without Format

Type        Var1
----      --------

Type1    -0.005855
Type2     0.012498
Type3    -0.047033

Example 5 - Report with Percent Format

Type       Var1
----     --------

Type1     ( 0.59%)
Type2       1.25%
Type3     ( 4.70%)

Example 5 - Report with Picture Format

Type       Var1
----     --------

Type1      -0.59%
Type2       1.25%
Type3      -4.70%
```

6 – DISPLAY DATETIMES

There aren't too many stored datetime formats but the picture format with **directives** can be used to describe any picture for your date, time, or datetime variable. This technique is especially useful for formatting SAS dates as DBMS dates for use in SQL queries and for special reporting specifications. In this example we want to create one datetime as the current datetime and another as an assigned value. Both need to be formatted in this odd combination with dashes and dots. We are using the option DATATYPE to indicate that directives will be used to create a template for the picture format. The directives here are the % indicators for the components of the datetime. Since the result is a picture, we must convert the original numeric datetime value to a character value using the PUT function.

Example 6 – Program

```
proc format;
      picture dtpic
                    other='%Y-%0m-%0d-%0H.%0M.%0S' (datatype=datetime)
                    ;
run;

data ex6;
     dt_beg = put(datetime(),dtpic.);
     dt_end = put('01JAN9999-00.00.00.000000'dt,dtpic.);
run;

proc print;
run;
```

Example 6 - Output

```
          Example 6 - Display Datetimes

Obs          dt_beg                   dt_end

 1      2003-11-03-18.01.05     9999-01-01-00.00.00
```

7 – CODE STYLES CONDITIONALLY

Suppose you are creating a report and you want to define a report style conditionally, depending on the value of the data. For example, you want to prepare an Excel spreadsheet with all the negative values in red. You could do this in PROC REPORT using a compute block with an IF statement and a CALL DEFINE. You would have to create a compute block for every variable. But you could also use PROC FORMAT to define the style as a format and then just reference the format when you define the variable. In this case we are not going to reference the format in a format statement, but rather in a style specification. Here's the example.

The format defines a range of data and the color associated with each range. Of course you could define many more ranges and many more colors if you like. You could also use it to define a different kind of style, like a font, for instance. Then the style option in the DEFINE statement just references the format and you don't have to use any compute blocks at all.

Example 7 – Program

```
proc format;
     value negfmt
           low-0 = 'red'
           other = 'black'
           ;
run;

ods listing close;
ods html body="&home/ex7a.xls";

proc report data=ex7a nowd split='~'
     style(report)=[background=white foreground=black]
     style(column)=[background=white foreground=black font_size=2]
     style(header)=[background=white foreground=blue font_size=2]
     ;
     columns code var1 var2 var3;

     define code  / 'Code';
     define var1  / format=4.0 style=[foreground=negfmt.] 'Var 1';
     define var2  / format=4.0 style=[foreground=negfmt.] 'Var 2';
     define var3  / format=4.0 style=[foreground=negfmt.] 'Var 3';
run;

ods html close;
ods listing;
```

Example 7 – Output

The Excel report we created looks like this.

Code	Var 1	Var 2	Var 3
ABC	123	456	789
DEF	**-321**	345	654
GHI	567	**-456**	986
JKL	945	345	**-75**
MNO	**-345**	**-567**	**-234**
PQR	934	49	89
STU	**-345**	**-945**	789
VWX	34	**-456**	**-567**

8 – EXTRACT RECORDS FROM A VERY LARGE DATASET

A very useful and clever way to use a format is to define the subset of a large dataset that you want to extract and read in the data using that format. It is similar to the first example in that it does define the data by groups. One group is the data that you want to keep, and the other group is the data to be discarded. But it is different in that it is used on input whereas formats are usually applied for output display.

The first part of the program uses a list of id values from a small file to identify the records that are to be extracted from the big file. ID1 is the variable we want to key on and it is used to create a format called MFMT. The CNTLIN option is used to store the format in FMTFILE. Note: The CNTLIN option of PROC FORMAT allows us to create a format from data in a SAS dataset. It requires that we create the variables FMTNAME, START, LABEL, and TYPE (for character data).

Then the big file is set in and we match the id variable there (LN_ID) to our key variable in the format MFMT. By using the subsetting IF, we instruct the program to keep only those records that match the values defined in the format. So that means we will only read in the records that match to the values from the list of ID1 values.

This technique is especially useful if the big dataset is not sorted because it does not require a sort and therefore saves the resources. It is also especially effective when the small dataset is significantly smaller than the large one.

Example 8 - Program

```
*-------------------------------------------------*;
*    Use id1 from small file to create a format   *;
*-------------------------------------------------*;
data keydata;
     set home.dataset8(keep=id1);
run;

data fmtfile;
     set keydata end=eof;
     length start $10
            label $10
            key $10
            ;
     key=id1;
     start=key;

     fmtname = 'mfmt';
     type = 'C';
     label = 'Match';

     output;

     if eof then do;
        start = 'other';
        label = 'No Match';
        output;
     end;
run;

proc sort data=fmtfile nodupkey;
     by key;
run;

proc format cntlin=fmtfile;
run;
```

```
*-------------------------------------------------*;
*    Subset Big file using id1 format             *;
*-------------------------------------------------*;
data address;
     drop key keyfmt;
     length key $10
            keyfmt $10
            ;
     set appldir.address;
     key = ln_id;
     keyfmt = put(key,$mfmt.);
     if keyfmt = 'Match';
run;

*-------------*;
*    Merge    *;
*-------------*;
proc sort data=home.dataset8
          out=dataset8;
     by id1;run;
proc sort data=address(rename=(ln_id=id1));
     by id1;run;
data newfile(label="Dataset8 with Addresses");
     merge dataset8(in=in8) address;
     by id1;
     if in8;
run;

proc contents data=newfile;
run;
```

Example 8 - Log

```
.
.
.
38
39         proc sort data=fmtfile nodupkey;
40              by key;
41         run;

NOTE: 1 observations with duplicate key values were deleted.
NOTE: There were 10001 observations read from the dataset WORK.FMTFILE.
NOTE: The data set WORK.FMTFILE has 10000 observations and 6 variables.
NOTE: PROCEDURE SORT used:
      real time           0.05 seconds
      cpu time            0.04 seconds

42
43         proc format cntlin=fmtfile;
NOTE: Format $MFMT has been output.
44         run;

NOTE: PROCEDURE FORMAT used:
      real time           1.29 seconds
      cpu time            0.19 seconds

NOTE: There were 10000 observations read from the dataset WORK.FMTFILE.
```

```

*--------------------------------------------------*;
*    Subset Big file using id1 format              *;
*--------------------------------------------------*;
data address;
     drop key keyfmt;
     length key $10
            keyfmt $10
            ;
     set appldir.address;
     key = ln_id;
     keyfmt = put(key,$mfmt.);
     if keyfmt = 'Match';
run;
```

```
NOTE: There were 34410649 observations read from the dataset APPLDIR.ADDRESS.
NOTE: The data set WORK.ADDRESS has 9995 observations and 5 variables.
NOTE: DATA statement used:
      real time           3:11.01
      cpu time            1:37.85
```

CONCLUSION

PROC FORMAT is often a more efficient way of transforming data than using DATA step code. Some of the advantages to using PROC FORMAT for data manipulation are:

- It is easier to code, read, and maintain.
- It can be stored permanently.
- It can be stored separately and changed without requiring program changes.
- Values can be specified as discrete values or ranges.
- Ranges can be explicit or implicit (LOW, HIGH, OTHER)
- Formatted lookups use a binary search.
- The PUT function with a format is more CPU efficient than a MERGE statement.
- Using one PUT function is more efficient than executing multiple IF statements.
- The entire format is loaded into memory.

These examples are just a few of the many applications of PROC FORMAT to input, manipulate, and output data. With some thought and creativity, it can be used to manage data far beyond the scope of just an attractive display. I encourage you to think of PROC FORMAT more often and to devise even more interesting applications.

CONTACT INFORMATION

The author can be reached at Lois831@hotmail.com.