

THE POWER OF PROC FORMAT

Jonas V. Bilenas, Chase Manhattan Bank, New York, NY

ABSTRACT

The FORMAT procedure in SAS® is a very powerful and productive tool. Yet many beginning programmers rarely make use of them. The FORMAT procedure provides a convenient way to do a table lookup in SAS. User-generated SAS Formats can be used to assign descriptive labels to data values, create new variables and find unexpected values. PROC FORMAT can also be used to generate data extracts and to merge data sets. This paper will provide an introductory look at PROC FORMAT for the beginning user and provide sample code (ver 6.12) that will illustrate the power of PROC FORMAT in a number of applications.

INTRODUCTION

PROC FORMAT is a procedure that creates mappings of data values into data labels. The user defined FORMAT mapping is independent of a SAS DATASET and variables and must be explicitly assigned in a subsequent DATASTEP and/or PROC.

PROC FORMAT can be viewed as a table lookup allowing 1-to-1 mapping or many-to-1 mapping. An example of a 1-to-1 mapping is the mapping of approve/decline codes used to process credit applications. In the consumer credit card industry, application processing decisions are often abbreviated. A simple example is;

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

If we have many approval codes and many decline codes, these can be mapped or assigned with a many-to-1 mapping. For example;

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

PROC FORMAT will not allow 1-to-many or many-to-many mappings. This is a good feature of PROC FORMAT since we don't want data values to take on more than one label. When using a DATASTEP and IF-THEN-ELSE logic to assign labels, the SAS LOG will not indicate if you have data values pointing to more than one label. Using PROC FORMAT instead of IF-THEN-ELSE code will provide an extra quality control check of your assignments of labels to data values.

Let's look at a few sample problems and see how PROC FORMAT can be used to generate more efficient code. These examples come from the consumer credit card industry but the concepts have application to many fields.

AN APPLICATION THAT ASSIGNS VALUES WITHOUT USING PROC FORMAT

When credit bureau data on individuals are requested, a generic credit bureau score can also be purchased. This score ranks predicted risk for the individual with higher scores being less risky than lower scores. One such score has integer values from 370 to 870 with missing scores assigned to values outside this range.

We wish to run a frequency distribution on individuals with scores less than 671 and those above 670. A beginning programmer would often handle this by creating another DATASET where a new variable is generated to assign membership into low scores, high scores and missing groups. A PROC FREQ is then submitted to get the desired frequency distribution.

```
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;
```

The results from the above code are shown in Figure 1.0 on the following page.

The SAS code above did the job, but it required that a new DATA-Step be created and the label 'unscored' was truncated to 'unsc'.

THE SAME VALUE ASSIGNMENT PROBLEM SOLVED USING PROC FORMAT

An alternative approach entails using a user-defined SAS Format. This saves some processing time and resources. Below (next page) is the SAS Code with

Figure 1.0

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

the solution to same problem but using PROC FORMAT.

The example here is of the simple value replacement variety of SAS Format. We will discuss how to set up a Format that assigns values to ranges in a subsequent section.

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

The SAS Code returns the output shown in Figure 2.0:

Some observations about this result are:

1. The name of the format does not have to be the name of the variable that it will be assigned to.
2. The assignment of the FORMAT occurs in the PROC with a FORMAT statement.
3. The format definition ends with the ';' on a new line. This is just my preference but I find it eas-

ier to debug PROC FORMAT code especially if I add more values to the format later on.

4. The '**unscored**' label now appears without truncation. When assigning a character label in a dataset, the length of the first evaluation of the label will be applied to all labels.

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 labels will be the original values. Here is an example:

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

The output of this code is shown in Figure 3.0 below:

Figure 2.0

SCORE	Cumulative		Cumulative	
	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

Figure 3.0

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

GENERATING NEW VARIABLES WITH PROC FORMAT AND A VALUE REPLACEMENT 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. In this example we wish to assign expected delinquency rates for given score ranges for a given portfolio.

```
proc format;
  value edr low-159 = '53.4'
           160-169 = '39.3'
           170-179 = '32.3'
           180-high = '25.8'
;
data stuff;
  set cb2;
  edr=input(put(score,edr.),4.1);
run;
```

With the above code a new variable called edr is generated from the call of the format in the PUT function. PUT always return a character, so the INPUT function was used to convert the variable to numeric since we required that the edr variable be a numeric variable.

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

We note the following observations about this usage:

1. If values are character, use a format name that begins with a '\$'.
2. Note that you can create many formats with a single PROC FORMAT statement. Just start each new FORMAT with a VALUE or PICTURE statement and end each definition with a ';'.

SPECIFYING RANGES IN PROC FORMAT

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

1. Ranges can be constant values or values separated by commas:
 - 'a', 'b', 'c'
 - 1,22,43
2. Ranges can include intervals such as:
 - <lower> - <higher>. means that the interval includes both endpoints.
 - <lower> <- <higher>. means that the interval includes higher endpoint, but not the lower one.
 - <lower> - < <higher> means that the interval includes lower endpoint, but not the higher one.
 - <lower> <- < <higher> means that the interval does not include either endpoint.
3. The numeric "." and character " " missing values can be individually assigned values.
4. Ranges can be specified with special keywords:

LOW	From the least (most negative) possible number.
HIGH	To the largest (positive) possible number.
OTHER	All other numbers not otherwise specified.
5. The LOW keyword does not format missing values.
6. The OTHER keyword does include missing values unless accounted for with a '.' or ''.

USING PROC FORMAT FOR DATA MERGES

SAS now includes an INDEX option on data sets to merge DATASETS that are not sorted, but PROC FORMAT also offers a method of merging large data sets (up to 100,000 records) to very large (millions of records) unsorted DATASET or flat file.

The method first builds a user defined format from a DATASTEP using a CNTLIN= option. The smaller file must not have any duplicates of the key variable used for matching. The DATASET created must have these elements:

- 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 **DATASET** or larger flat file. Here is an example of code:

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

data fmt (rename=(seqnum=start));
  retain fmtname '$key'
  type 'C'
  label 'Y';
  set temp;

proc format cntlin=fmt;
run;
data _null_;
  infile bigfile;
  file extract;
  if put(seqnum,$key.)='Y'
    then put _infile_ ;
run;
```

We observe the following about this code:

- The sort of the small **DATASET** was done to ensure no duplicates of the key field **SEGNUM**.
- This code extracted the entire record of the large flat file if there was a match in the key field and put that record into a file with a filename of **extract**.
- A match merge extract could work just as well using a SAS Dataset by modifying the **DATA-Step** as follows:

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

SAS PICTURE FORMATS

SAS **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 (i.e., %)
- Prefixes.
- Truncation or rounding of numbers.

One example of using **PICTURE FORMATS** is to truncate numbers that represents dates from **YYMMDD** display to **YY/MM**.

```
proc format;
  picture dt 0-991231='11/11'
    (multiplier=.01)
  ;
```

The '11/11' specifies that all leading zeros will be displayed. If the label was '01/11' then 001213 would be displayed as 0/12 instead of 00/12.

The **MULTIPLIER** option truncates the **DD** portion of the date when printing.

Another example of **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%).

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

SAVING FORMATS AND USING THEM LATER:

Often you may want to permanently save your formats, to use in other code or to be used by other users. This saves having to regenerate them at each point of usage. To do this, use the **LIBRARY=** option on the **PROC FORMAT** statement.

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

The **LIBRARY=** reference is a SAS libname and **PROC FORMAT** then places the generated format in a **CATALOG** with the name "FORMATS" in the libname specified. If the catalog does not already exist, SAS will create it.

To use the saved format in a subsequent program without having to enter the **FORMAT** code, specify a **LIBNAME** of **LIBRARY** referencing the library where the **FORMATS** catalog resides that contains the stored formats you want to use.

```
libname LIBRARY "... <def'n> ...";
```

Please note that this libname reference has to be LIBRARY.

MODIFYING VALUE REPLACEMENT FORMATS

You can edit the SAS Format, but not in the form it assumes in the Format Catalog. You must first copy the format to a SAS Dataset. This you do using PROC FORMAT with the **CNTLOUT=** option.

```
proc format library=libref
              cntlout=sas dataset;
  select entry;
run;
```

Edit the SAS Dataset using, for example, PROC FSEDIT or PROC FSVIEW. Then regenerate the User Format from the updated SAS Dataset using PROC FORMAT with the CTLIN= option.

PRINTING A SAVED FORMAT LIBRARY

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

```
libname libraray ...
proc format library=library fmtlib;
run;
```

OTHER FORMAT OPTIONS AND REQUIREMENTS WHEN USING SAS VERSION 6.12

Some other things you need to know when you define SAS Formats are:

- User-defined SAS Format names must be eight characters or less
- User defined SAS Format names cannot end in a number.
- You can define SAS Value replacement Formats to be applied to character values, as opposed to numeric values. The names of these must begin with a '\$' This is part of the eight characters available for the format's name. Thus the actual format name for a character format is seven characters or less.
- PROC FORMAT has some options whereby one may specify the length of the label in a value replacement format. These are: MIN=, MAX=, and DEFAULT=
- INFORMATS can be created in PROC FORMAT with the INVALUE statement. My particular code applications happen to use only user defined SAS Formats -- i.e. using only the VALUE and PICTURE statements -- and hence the subject has not been covered in these notes..

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.

CONTACT INFORMATION

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

Jonas V. Bilenas
Chase Manhattan Bank
Decision Science, Experimental Design
55 Water Street, 1618
New York NY 10041
Email: Jonas.Bilenas@Chase.com

ACKNOWLEDGMENTS:

SAS is a registered trademark of the SAS Institute, Inc. of Cary, North Carolina.