

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 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 maps 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 1 to 1 mapping is the mapping of approve/decline codes. 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 when assigning labels the SAS LOG will not indicate if you have data values pointing to more than one label.

Lets look at PROC FORMAT and applications using sample SAS code. These examples come from the consumer credit card industry but the concepts have application to many fields.

AN APPLICATION OF ASSIGNING 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;
```

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'.

SAME PROBLEM USING PROC FORMAT CODE

Using a user defined FORMAT saves some processing time and resources. The same problem solved with a PROC FORMAT is listed here. We will discuss how to set up ranges in a subsequent section.

```
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:

SCORE	Frequency	Percent	Cumulative Frequency	Cumulative 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. Name of the format does not have to be the name of the variable that it will be assigned to.
2. Assignment of the FORMAT occurs in the PROC with a FORMAT statement.
3. 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 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 look 370-870 = '370-870'
;
proc format data=cb;
  tables score;
  format score looky.;
run;
```

Output is as follows:

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

GENERATING NEW VARIABLES WITH PROC FORMAT

New variables can be assigned within a DATASTEP 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 sushi;
  set sashimi;
  edr=input(put(score,edr.),4.1);
```

With the above code a new variable called edr is generated from the call of the format edr. 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' ;
```

Some observations to make:

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 FORMAT 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. Interval includes both endpoints.
 - lower <- higher. Interval includes higher endpoint.
 - lower - < higher. Interval includes lower endpoint.
 - lower <- < higher. 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.
5. 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 CONTLIN= 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;
```

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

Some observations:

- The sort of the small DATASET was done to ensure no duplicates of the key field. If it is known that no duplicates exist, the small data set does not have to be sorted.
- This code extracted the entire record of the large flat file if there was a match in the key field and put the record in file with a filename of extract. A match merge extract could of worked off a SAS DATASET by modifying the DATASTEP as follows:

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

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 (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 YYMM. This code also embeds a '/' in the display between the YY and MM:

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

The '11/11' specifies all leading zeros 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

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 libarray ...
```

You can edit the format by first converting it into a SAS DATASET and then edit with PROC FSEDIT. When the changes are made, convert back to a user FORMAT with the CTLIN= option.

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

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

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

OTHER FORMAT REQUIREMENTS (VER 6.12)

- Format names must be 7 characters or less and cannot end in a number.
- Formats applied to character values must begin with a '\$'.
- Format has some options to specify the length of the label; MIN=, MAX=, DEFAULT=
- INFORMATS can be created in PROC FORMAT with the INVALUE statement. My code application stick with VALUE and PICTURE statements.

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
Risk Management, Experimental Design
55 Water Street, 1618
New York NY
Email: Jonas.Bilenas@Chase.com