Paper 189-2008

Get your hands dirty using the FORMAT Procedure

Marge Scerbo, National Study Center, UMB

Abstract

SAS® programming avails coders multiple ways to achieve a goal. Many programmers have learned to use DATA step statements to add labels to values and/or to group values. While this code works fine, it is often cumbersome, time consuming to construct, and hard to maintain.

The same goals may be obtained by using PROC FORMAT. This procedure can often add a great deal of efficiency to your programming, but it may be difficult to get started. Join this hands on workshop to use your hands to learn the basics of the FORMAT procedure. The best way to learn is to use.

Once the basics have been covered, we may try our 'hand' at other uses for the procedure.

Introduction

The FORMAT procedure can be a programmer's best friend once a programmer is comfortable with the basics. No procedure can replace the power of the SAS DATA step, but learning to use procedures in the right place and at the right time may provide new and efficient methods to the same end. There are many uses for Proc FORMAT including the ability to add labels to data values. This paper and workshop will specifically cover the VALUE statement to label values or groups of values.

Proc FORMAT is part of the Base SAS language. It can create formats that in turn can be used by either another DATA or PROC step.

DATA

Throughout this presentation, we will be using one dataset containing information on patient visits. Here is a sample of the fields and the values stored:

patientid	dx1	dx2	units	procedure	provider_type	gender	age
J00013610	75679		1	99214	20	M	11
J00019800	V242		1	W9091	20	M	14
J00025070	64293		1	W9091	20	M	16
J00030350	38101	5589	1	92567	20	F	24
J00057700	V222		1	W9091	20	F	36
J00082420	4770		1	99212	20	F	48
J00127900	V221		1	99213	36	M	58
J00169580	5959	9390	1	99212	20	M	68
J00196200	7793		1	99211	20	F	78
J00212950	64223	6442	1	90780	20	M	86

We will also use a table of diagnosis codes and descriptions.

DATA Step Variable Labels

First, it is important to clearly understand the difference between a variable label and a value label.

- By using a LABEL statement in either a DATA or PROC step, a variable label can be attached to a variable. This label adds descriptive information to the name of the variable, for example the variable *SSN* labeled as 'Social Security Number'.
- A value label for a variable is usually used for one of two purposes:
  - to describe the codes or other values of the variable
  - to group values for study

Grouping is very important in many studies:

- Many analyses are reported by groups, not by individual values
- Groups often show patterns that individual analyses cannot
- In frequency studies where there are continuous numbers or a large number of values to be reported, grouping limits the amount of output

Beginning SAS programmers learn to add descriptive labels to their variable values by using IF/THEN/ELSE statements and creating new variables. This process is completely correct but may be inefficient if the new variable is not needed on a regular basis and/or the dataset is very large. Often, these new variables are needed for one-time reports and not for any analytic process.

In the example below, IF/THEN/ELSE statements are used to create values for the field, *gender*, by creating a new variable called *gendername*. Note that the IF statement is preceded by a LENGTH statement to define the new variable, *gendername*. This length is equal to the longest possible value of the variable. The IF statement queries the value of *gender* until a match for the variable value is found. Since the variable is a character variable, note that the values are placed in quotes:

```
*data step labels;
data example1;
      set how.physician;
      length gendername $6;
      if gender = 'M'      then gendername = 'Male';
      else if gender = 'F' then gendername = 'Female';
run;
proc freq data = example1;
      tables gendername / nocum;
run;
```

In this example, a new dataset, *example1*, is created with a new variable, *gendername*. This process produces a clean LOG, and the output requested, but it is always important to assess efficiency, especially when large datasets are involved.

The output created from this DATA and PROC step is shown below:

Gender	Frequency	Percent
Male	11062	33.00
Female	22047	67.00

This DATA step process is correct, but Proc FORMAT can also provide the same output.

PROC FORMAT Syntax

There are a number of rules for creating and using formats. As is the case in most SAS code, the variable type is extremely important and, as such, defines which format type should be used:

- To format a character variable, create a character format
- To format a numeric variable, create a numeric format

Format names are defined in the VALUE statement. The rules for format names are:

- Character formats must begin with a $
- Numeric formats cannot begin with a $
- A valid SAS format name can be up to 32 characters in length (the $ counts as one of the 32 characters in a character format name)
- A format name must start with a letter or underscore (the first character after the $ in a character format name)

- Format names can contain only letters, numbers, and underscores
- A format name cannot end in a number
- A format name cannot be the name of a SAS function

Variable values on the left side of the equal sign can be:

- An individual value
- Multiple values separated by commas
- A range of values separated by a dash
- Combinations of individual values, multiple values, and ranges
- Values should not be repeated or overlapped
- Numeric values cannot be not enclosed in quotes
- Character values can be enclosed in quotes
- The keywords LOW and HIGH can be used to define the lowest values and highest values of the variable when the format is associated with a variable
- The keyword OTHER can be used to capture other non-missing values
- The keywords LOW, HIGH, and OTHER do not categorize missing values

The rules for the values on the right side of the equal sign include:

- A format label may be up to 256 characters in length
- Quotation marks should be placed around format labels

Multiple formats can be created with one Proc FORMAT by using multiple VALUE statements. Remember, variable value(s) on the left side of the equal sign print as the character value(s) on the right.

Formatted values are always character, regardless of whether the variable and the format is character or numeric. These formatted values cannot be used as numbers, but the internal value is unchanged, so the original variable value can still be used in procedures.

You can never go wrong by first running a frequency on the raw values and then on the corresponding formatted values to see if they make sense.

Now that the rules are laid out, let's resolve the previous example using a Proc FORMAT rather than a DATA step:

```
*Character format;
proc format;
       value $sex
              M = 'Male'
              F = 'Female';
run;
proc freq data = how.physician;
       tables gender / nocum;
       format gender $sex.;
run;
```

The first step is to create a new format:

- In this case a temporary format is created. If the format is temporary, it must be created in the same session (and prior to) any PROC or DATA step accessing the format:
  ```
  proc format;
  ```
- The VALUE statement names the format to be created, in this case a character format called *$sex*. Note that this format does NOT end in a period and is NOT followed by a semicolon:
  ```
  value $sex
  ```
- Now the variable values and their labels are listed. Note that the labels are enclosed in quotes and that the last variable description is followed by a semicolon:
  ```
  M = 'Male'
  /* many more here if needed*/
  F = 'Female' ;
  ```
- Remember when dealing with character values, that the match must be exact. If the value to be formatted is uppercase, make sure the format is also uppercase!

Numeric Formats

The creation of user defined numeric formats is very similar to the character format creation. There are several important issues to remember:

- The format name does not begin with a dollar sign
- The labels created are character, not numeric, and therefore can not be used in calculations
- The internal value of the number is not changed

In the following example, we will group the units of service (*units*). We can do this using DATA step IF/THEN/ELSE statements. Note that the variable values are not placed in quotes:

```
*numeric data step labels;
data example2;
       set how.physician;
       length unitcls $8;
       if units le 1       then unitcls = '1';
       else if units le 5  then unitcls = '2 to 5';
       else if units le 10 then unitcls = '6 to 10';
       else unitcls = 'GT 10';
run;
proc freq data = example2;
       tables unitcls / nocum;
run;
```

Results in this output:

Unitcls	Frequency	Percent
1	32533	98.26
2 to 5	277	0.84
6 to 10	202	0.61
GT 10	97	0.30

We can accomplish the same value labeling using Proc FORMAT. Note that neither a DATA step nor a new dataset is required in order to produce the same report. In the format below, the keywords LOW and HIGH are used:

```
*numeric format;
proc format;
       value unitfmt
              low - 1 = '1'
              2 - 5   = '2 to 5'
              6 - 10  = '6 to 10'
              11-high = 'GT 10';
run;
proc freq data = how.physician;
       tables units / nocum;
       format units unitfmt.;
run;
```

In this example, we created a numeric format called *unitfmt* and used this format in the Proc FREQ of the units of service. The resulting table is the same as the DATA step process.

Age Formats

Sometimes the study of a field or fields may require multiple views in order to obtain a reasonable and/or useful result. Age is a good example of that type of analytic field. Health care usage is often studied within different categories of age. It makes sense to group age in different ways to identify which best fits the needs of that particular study.

```
*different groupings;
data example3;
       set how.physician;
       length age1 age2 $12;
       if age le 20 then age1 = 'Children';
       else if age le 65 then age1 = 'Working Age';
       else age1 = 'Retired';

       if age le 5 then age2 = 'Preschool';
       else if age le 18 then age2 = 'Student';
       else age2 = 'Adult';
run;
proc freq data = ages;
       tables age1 age2 / nocum;
run;
```

Producing these listings:

age1	Frequency	Percent
Children	2642	8.00
Working Age	28092	85.00
Retired	2375	7.00

age2	Frequency	Percent
Preschool	587	2.00
Student	1743	5.00
Adult	30779	93.00

Now we will accomplish the same results using Proc FORMAT.

```
*different groupings;
proc format;
       value agea
              low - 20  = 'Children'
              21- 65    = 'Working Age'
              66 - high = 'Retired'    ;
       value ageb
              low - 5   = 'Preschool'
```

```
                5 - 18    = 'Student'
                19 - high = 'Adult' ;
run;

proc freq data = how.physician;
       tables age / nocum;
       format age agea.;
       title 'Age Grouping 1';
run;

proc freq data = how.physician;
       tables age / nocum;
       format age ageb.;
       title 'Age Grouping 2';
run;
```

Formats for Crosstabs

One descriptive mechanism for studying data is to create cross tabulation tables to see where the data values intersect. Creating crosstabs with Proc FREQ can produce volumes of results, often obscuring patterns. This is particularly a problem with continuous data. There may be times where creating formats to study these crossing might prove efficient.

The following code will produce a table with little (or maybe too much) information:

```
proc freq data = how.physician;
       tables units * age/ nopercent norow nocol nocum;
run;
```

We may be better served by first creating two separate formats, one for units and one for age. Note that one Proc FORMAT can create any number of formats. Using these formats, we will then run the same frequency procedure to create output that is easier to read.

```
proc format;
          value agea
                low - 20  = 'Children'
                21 - 65   = 'Working Age'
                66 - high = 'Retired'    ;
          value unitfmt
                low - 1 = '1'
                2 - 5   = '2 to 5'
                6 - 10  = '6 to 10'
                11-high = 'GT 10';
run;

proc freq data = how.physician;
       tables units * age / nopercent norow nocol nocum;
       format age agea. units unitfmt.;
run;
```

The resulting table is quite specific. This type of output provides a pretty clear view for the researcher.

Table of UNITS by age				
UNITS(Units)	age			Total
	Children	Working Age	Retired	
1	2602	27599	2332	32533
2 to 5	24	241	12	277
6 to 10	9	166	27	202
GT 10	7	86	4	97
Total	2642	28092	2375	33109

Missing Data

When you use Proc FORMAT, take care when dealing with missing values. The keywords, HIGH, LOW, and OTHER, do not capture missing data. If the analytic requires precise tables that include missing values, include the missing values within the VALUE statement (' ' for character and . for numeric). Additionally, make sure to use the MISSING option (if available) correctly within the Proc step. Below, note that the FORMAT procedure creates a format for specialty that includes missing data. The Proc FREQ includes the MISSING option on the TABLES statement.

```
*non inclusive of missing values;
proc format;
      value $speca
            20 = 'PCP'
            23 = 'Pediatrics'
            32 = 'Internal Medicine'
            34 = 'OB/GYN'
            36 = 'Orthopedics' ;
run;

proc freq data = how.physician;
      tables provtype / nocum;
      format provtype $speca.;
run;
```

PROVTYPE	Frequency	Percent
PCP	31704	96.16
Pediatrics	361	1.09
Internal Medicine	195	0.59
OB/GYN	553	1.68
Orthopedics	158	0.48

Frequency missing = 138

```
*includes missing values;
proc format;
       value $specb
              20 = 'PCP'
              23 = 'Pediatrics'
              32 = 'Internal Medicine'
              34 = 'OB/GYN'
              36 = 'Orthopedics'
              '' = 'Missing';
run;

proc freq data = how.physician;
       tables provtype / nocum missing;
       format provtype $specb.;
run;
```

for this output:

PROVTYPE	Frequency	Percent
Missing	138	0.42
PCP	31704	95.76
Pediatrics	361	1.09
Internal Medicine	195	0.59
OB/GYN	553	1.67
Orthopedics	158	0.48

The percentages in this table have changed by including the missing values.

Multiple Methods

SAS often provides multiple solutions to the same problem. For example, the physician's office needs reports based on diagnoses. These are stored as diagnosis codes in variables called *dx1* to *dx3*. The reports are to include the description of the diagnosis rather than the code. IF/THEN/ELSE statements can be written in a DATA step that queries each code and attaches the description based on that code. This process is very time consuming!

```
*First DATA step method;
data diagnoses;
       set how.physician;

       length diagnosis $50;
       if dx1 = '250' then diagnosis = 'DIABETES MELLITUS';
       else if dx1 = '2500'
              then diagnosis = 'DIABETES MELLITUS WO COMPLICATION';
       *add more here;
run;
```

```
proc print data = diagnoses;
       var dx1 diagnosis;
run;
```

Now assume there is an additional dataset called ICD9 that contains the diagnosis code (*icd*) and the description (*description*). We can accomplish the same task by merging the datasets:

```
*Second DATA Step method;
*first sort both datasets;
proc sort data = how.icd9 out = icd;
       by icd;
run;

proc sort data = how.physician out = phys;
       by dx1;
run;

*merge both datasets by the key variable;
*note that the variable icd is renamed to dx1;
data diagnoses;
       merge phys (in = inphys) icd (rename = (icd = dx1));
       by dx1;
       if inphys = 1;
run;

proc freq data = diagnoses;
       tables description / nocum;
run;
```

This process will create the correct reports, but sorting and merging large datasets can be inefficient.

You can also use Proc FORMAT to create a label for each diagnosis code. This process will produce the same output and can be easily updated to create the reports as new diagnoses. No new dataset is created. Note that we are creating a character format called *$dxs*.

```
proc format;
       value $dxs
              250  = 'DIABETES MELLITUS';
              2500 = 'DIABETES MELLITUS WO COMPLICATION'
              /*add more here*/   ;
run;

proc freq data = how.physician;
       tables dx1 / nocum;
       format dx1 $dxs.;
run;
```

There's another way to solve this problem and that is by creating a format from the player names dataset.

Datasets for Formatting

Using the CNTLIN option within a Proc FORMAT allows you to create a user-defined format from values stored in a SAS dataset. This is a very efficient method to add value labels from a 'lookup' dataset, rather than merging two datasets, a master and a lookup. The syntax needed to create this format-based dataset is very specific. It requires the following variables:

- start this variable contains the codes or the values to be formatted
- label this variable contains the descriptions or labels for the values
- fmtname this variable is the actual name of the format

In the example below, the ICD9 diagnosis codes and their descriptions are stored in a dataset called ICD9. This process creates a format from the ICD9 dataset with the diagnosis code (*icd*) becoming the **start** and the code's description (*description*) becoming the **label**. The new dataset ICDS contains these variables plus a new variable called *fmtname* that is equal to the value *$dxnames* for every observation. Note that the variable *fmtname* is always character and must be placed in quotes. This variable is initialized in the RETAIN statement and carried forward in the following observations. If a character format is being created, include the dollar sign within the quotes. Note also that the format name does not include the period!

Proc FORMAT then uses this dataset in the CNTLIN option:

```
*report 3;
data icds;
       retain fmtname "$dxnames";
       set how.icd9;
       start = icd;
       label = description;
       keep start label fmtname;
run;

proc format cntlin = icds;
run;
```

This is what a portion of the dataset FORMATNAMES looks like

start	label	fmtname
001	CHOLERA	$dxnames
0010	CHOLERA VIBRIO CHOLOREA	$dxnames
0011	CHOLERA VIBRIO CHOLOREA EL TOR	$dxnames
0019	CHOLERA UNSPEC	$dxnames
002	TYPHOID AND PARATYPHOID FEVERS	$dxnames

Now we can use this format in our frequency. Note we do not need to create a new dataset or new variable.

```
proc freq data = how.physician;
       tables dx1 / nocum;
       format dx1 $dxnames.;
run;
```

Creating New Variables

Though formats can avoid the necessity of creating new variables for performing analyses, there may be occasions when you do want to create a new variable. Formats can also be used to create new variables within a DATA step.

This can be accomplished by using the PUT function along with the format, be it a SAS or user-defined format. Remember that all format values are character, whether or not the variable is numeric.

The PUT function can create a new variable based on the value of an already existing variable and a format. In the following example, a new variable, *dxname*, is created using the format created in the previous example:

```
*Creating new fields from formats;
data addnames;
       set how.physician;
       length dxname $50;
       dxname = put(dx1,$dxnames.);
run;
```

Permanent Formats

Just as with datasets, you have the option of saving your formats permanently. Formats are not stored in datasets but rather in catalogs. By defining the library where the catalog is located and using the LIBRARY= statement, the formats can be stored permanently:

```
libname ourlib 'j:/shared/physician';
proc format library = ourlib;
       value $spec
              20 = 'PCP'
              23 = 'Pediatrics'
              32 = 'Internal Medicine'
              34 = 'OB/GYN'
              36 = 'Orthopedics' ;
       value $sex
              M = 'Male'
              F = 'Female' ;
run;
```

These formats are now stored in a permanent catalog. If they are placed on a network drive, all analysts can access those catalogs by simply using the correct LIBNAME statement. Remember these formats are available for use in both DATA and PROC steps as long as they are defined before use.

Adding the option FMTSEARCH causes SAS to search format catalogs in the order listed, until the desired member is found. Here is a simple example using a permanent format. Note that the catalog is identified in the LIBNAME statement prior to use and the possible locations of the format to be used are listed in the FMTSEARCH option:

```
libname medical 'j:/shared/records';
libname how 'j:/shared/physician';

options fmtsearch = (medical how);
proc freq data = how.physician;
       tables gender * provtype / nopercent norow nocol nocum;
       format gender $sex. provtype $spec.;

run;
```

One More Advanced Example

When you are working with large datasets, efficiency is extremely important. You can use what you learned in this session to write more efficient SAS code.

In this example, management would like a series of reports of patients who received a service in 2007. A frequency table of the patients' zip codes and counties will provide the practices with important information. The identification numbers of these patients (*idnum*) can be selected from the physician file (PHYSICIAN) for service year 2007. A large dataset (PATIENTS) that contains information on one million+ patients will also be used in this process. This file contains the same unique identifier (*idnum*).

To prepare this report, we could use a SAS DATA step MERGE, but the larger the number of observations the slower the process. Instead, we will use what we have learned to produce a format of those fan ids:

```
data pats07;
       retain fmtname "$patients" label "in2007" ;
       set how.physician;
       if year(servicedate) = 2007;
       start = idnum;
       keep start label fmtname;
run;
proc format library = ourlib cntlin = pats07;
run;

data report4;
       set how.patients;
       where put(idnum,$patients.) = 'in2007';
run;
proc freq data = report4;
       tables zipcode county / nocum;
       title 'Patient Report 4';
run;

*OR combined steps;
proc freq data = how.patients;
       where put(idnum,$patients.) = 'in2007';
       tables zipcode county / nocum;
       title 'Patient Report 4';
run;
```

This process subsets the data simply and efficiently. When you use this method, make sure you have a pretty good idea what to expect from the output.

A Few Additional Notes

If you receive a dataset where variables have been stored with user-defined formats but no format catalog, you can access the dataset by 'turning off' the format check:

```
options nofmterr;
```

If you receive a format catalog but no definitions for what is actually stored in the catalog, you can print out the contents by using the FMTLIB option of the procedure:

```
proc format fmtlib;
run;
```

When you are working with character data, it is very important to know everything about the data. If you are formatting and/or matching, using the function UPCASE or LOWCASE will make the manipulation one step easier. You can also used mixed case in formats:

```
proc format;
       value $address
              street, STREET, ST, st = 'Street';
run;
```

If you are formatting a character field that contains a single quote, make sure to enclose the entire value in double quotes:

```
TOB = "Thomas O'Brien"
```

Conclusion

Again, the FORMAT procedure can be extremely useful when dealing with data. Remember to cross check your values by first running a Proc FREQ or a similar procedure and then running the procedure again with formatted values.

Contact Information

For more information contact:

Marge Scerbo
National Study Center
701 W. Pratt, Room 554
Baltimore, MD 21201
Email: mscerbo@som.umaryland.edu