SESUG Paper 134-2017

Format-o-matic: Using Formats To Merge Data From Multiple Sources

Marcus Maher, Ipsos Public Affairs; Joe Matise, NORC at the University of Chicago

ABSTRACT

User-defined formats are often the best way to merge a single value onto a larger dataset, but they can seem overly complicated for the novice programmer to use, particularly when remembering all of the specific variable names and details like the ‘OTHER’ row.

We first explain the use case for user-defined formats and some of the considerations to keep in mind when using them. Then, we present a single macro that creates a user-defined format from an already existing dataset, with parameters for all of the commonly used options and some of the less common ones, written for maximum flexibility. The macro is intended to be able to be used by novice programmers without complete knowledge of the workings of the process, but the advanced options make it appropriate for any level of programmer.

This presentation does not require any understanding of user-defined formats, or even SAS® formats at all. The intended audience is novice and intermediate level programmers, as well as anyone interested in an off-the-shelf user-defined format macro.

INTRODUCTION

We have trained many people with a wide variety of backgrounds to program in SAS. For those who have not had formal training in programming or mathematics, we often see beginner and intermediate programmers struggle through using user defined formats, if they even attempt to do so. The aversion to formats even manifests itself among more advanced programmers who find aspects of formats cumbersome. We would like to briefly reinforce the power of SAS formats, demonstrate some basic capabilities, and present a macro for generating format with the goal of lowering the barriers that many have to using them.

THE POWER OF SAS FORMATS

SAS formats take a domain (the data stored in a variable) and map it to a range (the desired output). These mappings can be one-to-one, many-to-one, or one-to-many. SAS formats are first seen as a tool to make output more attractive or meaningful, but they also are a key tool for efficient programming. Format look-ups are extremely fast, and an appropriate use of a format can eliminate expensive merges and ‘if-then’ or ‘select’ constructions.

In particular, we want to target merges, where we have a large main table which needs fields added from a small look-up table. A sort and a merge (or a join) can be a time-consuming I/O step. However, if it’s feasible to create formats from the lookup table that I/O step can, in many cases, be avoided, as one can apply the format when they are ready to use it in a PROC or later DATA step. If the main dataset is not trivial in size this can save substantial amounts of run time.

USING SAS FORMATS

We will now walk through a simple example for each type of mapping: one-to-one, many-to-one, and one-to-many.

ONE-TO-ONE MAPPINGS

The most common user defined formats supply some sort of description to an underlying code. For example, define a numeric variable that has two values, 1 and 2, where 1 means ‘Yes’ and 2 means ‘No.’ This mapping can be defined using a format named `yesnof.` as follows:

```
proc format;
   value yesnof
         1='Yes'
         2='No'
   ;
quit;
```

This format can then be called in a PROC FREQ statement to produce a table that has the formatted values:

```
proc freq data=have;
   table Qyesno;
   format Qyesno yesnof.;
run;
```

This produces the following output:

Qyesno	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Yes	122	48.80	122	48.80
No	128	51.20	250	100.00

Table 1. Sample table with SAS format applied to perform data point labelling

The 1 and the 2 are replaced with "Yes" and "No", and otherwise the table is identical to the table without the format. Note that the dataset at this point has not been changed permanently: because the format was applied in the PROC, the original dataset still has no format on the variable. This could be changed, either by applying the format to the variable in the dataset:

```
proc datasets lib=work;
  modify have;
  format Qyesno yesnof.;
  run;
quit;
```

in which case the underlying variable still has a value of 1 or 2, but it is displayed both when the dataset is viewed, and when most reporting procedures are run on it, with "Yes" and "No". Finally, the dataset could have a new variable created that stores the value of the format, like so:

```
data want;
 set have;
 Qyesno_c = put(Qyesno,yesnof.);
run;
```

which is the only way to actually change the value itself.

This is more useful in cases where formats are being used as a way to merge data from one dataset to another, which we will explore later on.

MANY-TO-ONE MAPPINGS

SAS formats can do far more than simply labelling a single data value with a single value. For an example of a many-to-one mapping, consider formatting a variable containing age stored as a number of years. That variable is commonly grouped into age bands. Rather than writing a recode to map many values down to one, it can be done with a format:

```
proc format;
   value agef
         0-17='Under 18'
         18-24='18-24'
         25-34='25-34'
         35-54='35-54'
         55-64='55-64'
         65-120='65+'
   ;
quit;
```

Note that a numeric range is specified by the starting and ending values separated by a dash, with the dash meaning ‘through.’ This approach is very useful in data step programming with recodes, similar to that of the one to one mapping, as well as frequencies as below:

```
proc freq data=have;
    table age;
    format age agef.;
run;
```

```
      age  Frequency  Percent  Cumulative  Cumulative
                               Frequency     Percent

 Under 18         39    15.60          39       15.60

    18-24         20     8.00          59       23.60

    25-34         26    10.40          85       34.00

    35-54         56    22.40         141       56.40

    55-64         36    14.40         177       70.80

      65+         73    29.20         250      100.00
```

Table 2. Sample table with SAS format applied assigning a range to a value

ONE-TO-MANY MAPPINGS

One-to-many mappings showcase one of the most powerful features of SAS when producing hierarchical summaries. They enable one row of data to be mapped to multiple rows of a report: for example, mapping a store to both a region and to a division without separate indicator variables, or mapping a product to two or more unrelated product categories. This functionality is available in several SAS procedures, such as PROC MEANS and PROC TABULATE, generally those that use the CLASS statement to classify data.

One-to-many mappings are known as *multilabel* formats in SAS, and are created using that keyword in the VALUE statement. They can then be used in a similar manner to one-to-one or many-to-one formats in procedures that are compatible with them, with the addition of the MLF option to the CLASS statement.

As an example, consider a county with two hospitals, and each hospital has several clinics. To analyze the average number of patients per hour at the state, hospital, and clinic level, one option would be to create three variables and run a PROC MEANS with each of these variables on the CLASS and TYPES statement. If each clinic has a unique name, however, this can be accomplished with a single variable on the class statement and a multilabel format:

```
proc format;
      value clinhf (multilabel)
            1-5='County #1'
            1-3='Hospital #1'
            1=’Clinic 1'
            2='Clinic 2'
            3='Clinic 3'
            4-5='Hospital #2'
            4='Clinic 4'
            5='Clinic 5'
      ;
quit;
```

Notice the addition of ‘(multilabel)’ on the VALUE statement. This instructs PROC FORMAT to allow each data point to be mapped to more than one formatted value; without this, SAS would give an error when overlapping ranges were specified. The rest of the VALUE statement proceeds as normal with the exception that more than one formatted value has been designated for each data value. This multilabel format is then utilized in PROC MEANS as follows:

```
proc means data=have;
    class clinic_num/mlf;
    var patient_vol;
    format clinic_num clinhf.;
 run;
```

Adding the ‘MLF’ option to the class statement instructs PROC MEANS to utilize the format as a multilabel format. The output table is below:

Analysis Variable : patient_vol

clinic_num	N Obs	N	Mean	Std Dev	Minimum	Maximum
Clinic 1	**24**	24	44.5833333	29.8444033	3.0000000	96.0000000
Clinic 2	**24**	24	51.6250000	25.1133301	17.0000000	97.0000000
Clinic 3	**24**	24	55.8750000	23.2609254	5.0000000	97.0000000
Clinic 4	**24**	24	49.1250000	29.8056203	2.0000000	93.0000000
Clinic 5	**24**	24	43.1666667	28.9041394	5.0000000	98.0000000
County #1	**120**	120	48.8750000	27.4500501	2.0000000	98.0000000
Hospital #1	**72**	72	50.6944444	26.2698021	3.0000000	97.0000000
Hospital #2	**48**	48	46.1458333	29.1999633	2.0000000	98.0000000

Table 3. Sample table with SAS format applied to perform one-to-many mapping in PROC MEANS

Each individual clinic is reported out, followed by the summary rows for county and hospital. These are displayed in alphabetical order; for further explanation of how to ensure a preferred order is preserved, please see usage note on using NOTSORTED with PROC FORMAT in the reference section below.

Note that, as opposed to the case with one-to-one and many-to-one mappings, multilabel formats cannot be used to recode data in the data step. While they can be applied in the data step as a value label which later on a multilabel format-enabled procedure can correctly utilize, they will only display a single value when browsing the dataset or using a non-multilabel-enabled procedure, or when using the PUT function to assign a value to another variable. There are ways to accomplish a recode using a multilabel-enabled procedure, but in most use cases there are simpler options available.

We do not attempt to cover the entire breadth of options available using formats; SAS has a number of other very useful options that are well worth exploring, and we highly recommend users who are interested read some of our recommended reading in order to further expand their knowledge on this very powerful feature of SAS.

THE FORMAT-O-MATIC MACRO

Now that we have covered the three types of mappings that can be done using PROC FORMAT, we will introduce our macro to simplify the process of creating user defined formats. The macro takes a dataset that defines your mapping as input and, given the arguments to the macro, load a format to the default formats catalog of your specified library (with WORK as the default). We will explain how to use the macro and fill in the parameters and then walk through a simple example.

DEFINING THE MACRO

The macro has eleven parameters that control the format generation. They cover where the information for the mapping is stored, the details of the mapping, what the format name and type should be, which library's catalog it should be stored in, and provide optional arguments to do deduping and debugging. The specifics are as follows:

```
%macro formatomatic(data= ,
                    start= ,
                    end= ,
                    to= ,
                    default=%str( ),
                    mlf= ,
                    library= WORK,
                    fmtname= ,
                    type= ,
                    dedup= ,
                    debug=
                    );
```

- **Data** – This is a required argument. It should contain the name of the SAS dataset that contains the mapping for the format. It can include dataset options.
- **Start** – This is a required argument. It should contain what to convert from. Specifically, it should hold a variable or expression that evaluates to a value. For many-to-one mappings the starting point of the range of values you are mapping from.
- **End** – This is an optional argument and can be left blank or omitted. If you are performing a many-to-one mapping this is the ending point of the range of values you are mapping from
- **To** – This is a required argument. It should contain what to convert to. Specifically, it should hold a value, variable, or expression that evaluates to a value.
- **Default** – This is an optional argument and can be left blank or omitted. It should contain a value, variable, or expression that evaluates to a value that is the value to display when no match is found. If blank or omitted, a missing value will be the default.
- **MLF** – This is an optional argument and can be left blank or omitted. If left blank no action will be performed. If a value is placed here the format will be a multilabel format for many-to-one mappings.
- **Library** – This is set to your WORK library by default. It should contain a value, variable, or expression that evaluates to a value that specifies the library that the format will be stored in.
- **Fmtname** – This is a required argument. It should contain a value, variable, or expression that evaluates to a value that is the desired name of your SAS format. Normal rules defining legal SAS format names apply.
- **Type** – This is a required argument. This contains the type of the format. It should take a value of ‘C’ for character formats and ‘N’ for numeric formats.
- **Dedup** – This is an optional argument and can be left blank or omitted. If left blank no action will be performed. If a value is placed here the format dataset will be de-duplicated by START prior to creating the format.
- **Debug** – This is an optional argument and can be left blank or omitted. If left blank the macro will delete the interim dataset. If a value is placed here, the interim dataset will be preserved.

UNDER THE HOOD

The format-o-matic macro utilizes the CNTLIN option of PROC FORMAT, which allows data to be imported directly into the format without requiring the VALUE statements above. The macro is not optimized for maximum speed, but rather is intended to save programmer time and reduce code length.

The macro creates a dataset, mapping the options specified in the parameters to the appropriate values for a CNTLIN dataset. It then optionally dedups the dataset, and finally imports the dataset to PROC FORMAT using the CNTLIN option. Finally, it cleans up the temporary dataset created in the macro when the DEBUG option is not set.

USING THE FORMAT-O-MATIC MACRO – AN EXAMPLE

One common use case for the format-o-matic macro is to create a format from an already existing dataset that can then be used to define a relationship on another dataset and/or in a report. In that case, using format-o-matic allows the programmer to obtain a format without going through the effort of creating the CNTLIN dataset and importing it.

In the following example, the PREDICT column from the dataset SASHELP.CLASSFIT is appended to the dataset SASHELP.CLASS, using the NAME variable as the key. DATA, START, TO, FMTNAME, and TYPE parameters are used; excepting TYPE, which only needs to be specified if the format is not numeric, these options are the minimum required to be specified for every call to the macro.

```
%formatomatic(data=sashelp.classfit,
              start=name,
              to=predict,
              fmtname=$classpredict,
              type=C
             );
data class_fit;
  set sashelp.class;
  predict = put(name,$CLASSPREDICT.);
run;
```

The DATA parameter to the macro permits the use of dataset options, most commonly WHERE, as is seen here:

```
%formatomatic(data=sashelp.classfit(where=(sex='F')),
              start=name,
              to=predict,
              fmtname=$classpredict_female,
              type=C
             );
data class_fit;
  set sashelp.class;
  predict = put(name,$classpredict_female.);
run;
```

Both START and TO can take expressions that evaluate to a value, as seen here:

```
%formatomatic(data=sashelp.classfit,
              start=name,
              to=log(predict),
              fmtname=$classpredict_log,
              type=C
             );
data class_fit;
  set sashelp.class;
  predict = put(name,$classpredict_log.);
run;
```

Finally, we will demonstrate a more complex use of the macro using the SASHELP.CARS dataset. For purposes of this example, the goal is to determine quartiles of MSRP and summarize mpg_highway with the quartile as a class variable to show summary statistics for each quartile of MSRP. We begin by using PROC MEANS to calculate the quartiles and store them in a dataset for further use:

```
proc means data=sashelp.cars noprint;
   output out=quartiles
         min(msrp)=min
         q1(msrp)=q1
         median(msrp)=q2
         q3(msrp)=q3
         max(msrp)=max;
run;
```

This gives us a dataset of a single row that has all of our quartiles. We then reshape the dataset to generate our input dataset for the format-o-matic macro. We store the lower bound of the range for the quartile in the variable ‘lowval’ and the upper bound of the quartile in the variable highval. The text description that we are mapping to is stored in the variable ‘quartile’:

```
data quartile_fmt;
   set quartiles;
         lowval=min;
         highval=q1;
         quartile='MSRP in Bottom Quartile';
         output;

         lowval=q1;
         highval=q2;
         quartile='MSRP in Second Quartile';
         output;

         lowval=q2;
         highval=q3;
         quartile='MSRP in Third Quartile';
         output;

         lowval=q3;
         highval=max;
         quartile='MSRP in Top Quartile';
         output;
         keep lowval highval quartile;
run;
```

This produces a dataset like the following:

lowval	highval	quartile
10280.0	20329.5	MSRP in Bottom Quartile
20329.5	27635.0	MSRP in Second Quartile
27635.0	39215.0	MSRP in Third Quartile
39215.0	192465.0	MSRP in Top Quartile

Table 4. Input dataset for FORMAT-O-MATIC macro

Each row contains the lower and upper bounds of the range for each quartile along with text identifying the quartile that the range pertains to. At this point, all we need to do is decide what library to store our desired format in, along with what we want to name the format. Then we can proceed to generating our macro call:

```
%formatomatic( data=quartile_fmt,
               start=lowval,
               end=highval,
               to=quartile,
               fmtname=msrpquartf,
               type=N
             );
```

In addition to the parameters specified for the simpler example, this example specifies the END parameter, signaling to the macro that we are using a range rather than a one-to-one mapping. We also specify type=N, as this is a numeric format; in this case this is optional (as numeric is the default assumed by SAS when no $ is present and numeric values are provided for START) but it is helpful to specify for clarity.

After we run the macro, our format has been created and we are free to use it as we wish. In our case, we want to see summary statistics for highway MPG based on the quartile of the MSRP. This is accomplished in PROC MEANS by placing MSRP on the CLASS statement and using a FORMAT statement to apply this format to the MSRP variable:

```
proc means data=sashelp.cars;
   class msrp;
   var mpg_highway;
   format msrp msrpquartf.;
run;
```

Analysis Variable : MPG_Highway MPG (Highway)

MSRP	N Obs	N	Mean	Std Dev	Minimum	Maximum
MSRP in Bottom Quartile	**107**	107	31.9252336	6.4730223	18.0000000	66.0000000
MSRP in Second Quartile	**107**	107	27.2149533	4.7484417	17.0000000	51.0000000
MSRP in Third Quartile	**107**	107	24.8691589	3.1863036	17.0000000	30.0000000
MSRP in Top Quartile	**107**	107	23.3644860	3.9772763	12.0000000	30.0000000

Table 5. Table of Highway MPG classified by MSRP Quartile as defined by the format from the FORMAT-O-MATIC macro

While this process was not substantially simpler than importing the formats directly from the intermediate dataset, using a single macro to perform most formats is helpful for code consistency, and can avoid the need for the programmer to remember the names of all of the variables and options in the CNTLIN dataset.

CONCLUSION

Formats are not only a useful tool for neatly displaying numeric values on the screen, but are also very useful tools for combining data from multiple sources, performing recodes without dozens of if/then/else statements, and performing complex hierarchical summarizations in a single step. We hope that this macro enables programmers to use user-defined formats more often, and simplifies their frequent use, so that they can get more out of their SAS experience.

RECOMMENDED READING

Levin, Lois. 2005. "PROC FORMAT – Not Just Another Pretty Face." *Proceedings of SUGI 30*, 2005. Available at: http://www2.sas.com/proceedings/sugi30/001-30.pdf.

Li, Stan. 2011. "Some Useful Techniques of Proc Format." *Proceedings of PharmaSUG*, 2011. Available at: https://www.pharmasug.org/proceedings/2011/CC/PharmaSUG-2011-CC19.pdf.

Usage Note 12904: "How to use the MULTILABEL and NOTSORTED options with PROC FORMAT." SAS Support: Samples and SAS Notes, 2004. Available at http://support.sas.com/kb/12/904.html .

Cody, Ron. 2008. "Using Advanced Features of User-defined Formats and Informats." *Proceedings of SAS Global Forum,* 2008. Available at http://www2.sas.com/proceedings/forum2008/041-2008.pdf .

Source code and examples for the macro are available at https://github.com/snoopy369/SESUG-2017

CONTACT INFORMATION

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

Marcus Maher
Ipsos Public Affairs
Marcus.Maher@ipsos.com

Joe Matise
NORC at the University of Chicago
matisejoe@gmail.com