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

To use mathematical terminology SAS formats take your domain (the data in your variables) and map it to your range (your desired output). These mappings can be one-to-one, many-to-one, or one-to-many. SAS formats are often used to make output more attractive or meaningful. 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 unnecessary 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, starting with one-to-one mappings.

## One-To-One Mappings

The most common user defined formats supply some sort of description to an underlying code. For example, let’s assume that you have a numeric variable that has two values, ‘1’ and ‘2,’ where ‘1’ means ‘Yes’ and ‘2’ means ‘No.’ Generating this label in PROC FORMAT is very simple. We must pick a name for our format (our mapping), which we will choose yesnof. Then the syntax is 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**;

Here is the 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

## Many-To-One Mappings

SAS formats can do far more than simply labelling a single data value with a single label (one-to-one mappings). They can also perform many-to-one and one-to-many (for multi-label enabled procedures like PROC MEANS, PROC TABULATE, and PROC REPORT) mappings. For an example of a many-to-one mapping consider the situation where you have a person’s age in years. Let’s assume that you want to display that information grouped into age bands. Rather than writing a recode to map many values down to one, it can be done with a simple 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 as #-# with the ‘-‘ specifying ‘through.’ Again, for demonstration purposes we will run a frequency to display the results of the mapping. However, we will note that many-to-one mappings via formats are a very powerful tool for DATA step programming. Please see SAS conference papers included in the reference section of this paper for some helpful examples:

**proc** **freq** data=have;

table age;

format age agef.;

**run**;

| **age** | **Frequency** | **Percent** | **Cumulative Frequency** | **Cumulative 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 to perform many-to-one mapping

## One-To-Many Mappings

One-to-many mappings are useful for producing hierarchical summaries and can only be fully utilized in certain SAS procedures, as their one to many nature is only in effect when applied to variables on a CLASS statement. Let’s assume that we have a county with two hospitals and that each hospital has some number of clinics. If we wanted to analyze the average number of patients per hour at the state, hospital, and clinic level, we could 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, we can accomplish this with a single variable on the class statement and a multilabel format. The syntax is as follows:

**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 VALIE statement. This instructs PROC FORMAT to allow each data point to be mapped to more than one formatted value. Then notice that the rest of the VALUE statement proceeds as normal with the exception that we have designated more than one formatted value for each data value. You can utilize this in PROC MEANS as follows:

**proc** **means** data=have;

class clinic\_num/mlf;

var patient\_vol;

format clinic\_num clinhf.;

**run**;

You have to add the ‘MLF’ option to the class statement for 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

We can see above that each individual clinic gets a row, and additionally we have summaries at the hospital and county level.

# The FORMATO-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 interim datasets. If a value is placed here, interim datasets will be preserved.

## using the formato-matic macro – an example

Now we will demonstrate use of the macro using the sashelp.cars dataset. For purposes of our example, our goal is to determine quartiles of MSRP and run a means on 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 formatomatic 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. Table of input dataset for formatomatic 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

);

For the purposes of this example we need to provide arguments to six macro parameters. The first parameter we need to define is the ‘data’ parameter and the value for that in this example is our data set ‘quartile\_fmt’ which contains the mapping of MSRP to quartiles. The second parameter is ‘start’ and is the variable in the quartile\_fmt dataset that contains the lower bound of each of the ranges for the quantiles, in this case ‘lowval.’ The third parameter is ‘end’ and is the variable in the quartile\_fmt dataset contains the upper bound of each of the ranges for the quantiles, in this case, ‘highval.’ The fourth parameter is ‘to’ and is the variable in the quartile\_fmt dataset that contains description we are applying to each quartile, in this case, ‘quartile.’ The fifth parameter is ‘fmtname’ and should contain the name we want to give to the format. We chose ‘msrpquartf.’ The final parameter is ‘type’ which here should be ‘N’ since ‘msrp’ is a numeric variable.

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. We accomplish this 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 our FORMATOMATIC macro

As we can see, the mapping was carried out by the use of the FORMAT statement in PROC MEANS yielding a table with the results that we desired.

# Conclusion

Formats are a vital tool that simplify enhance the clarity of syntax and are usually more efficient than merges or if/then constructs if the goal is to add a small number of variables. We hope that this macro enables more people to either use user-defined formats more often or make their use easier for those who struggle through them, so that they can get more out of their SAS experience.

# References

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.

# Contact Information

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

Marcus Maher

Ipsos Public Affairs

Marcus.Maher@ipsos.com