

Advanced Uses of SAS Formats

Howard Levine, DynaMark, A Fair, Isaac Company, St. Paul, MN

1. Abstract

This paper will show how to use SAS formats for grouping observations and performing table lookups. It will also show how to use informats for redefining input values.

There will be a brief overview of SAS formats and Proc Format. Then the advanced topics will be explored. Additional items covered will be picture formats, permanent format libraries, and using SAS data sets in conjunction with Proc Format.

2. Overview

SAS formats are useful tools for displaying output and for general data processing applications. Formats are, in essence, specialized tables of data that are optimized for fast lookups performed in memory. The lookups can be for discrete values or ranges of values. Once a hit on the lookup is found, appropriate action is taken to transform the value to what the format directs it to be. Here is an overview of how formats are used and how to use them.

2.1. Purposes of Formats

Formats are used to transform values in a variable to something else as the data is being read for a particular purpose.

2.1.1. Formatting Display of Output

When using any procedure that produces print output, a format can be used to control what the output looks like. Items that can be controlled are number of columns displayed, how date values are presented, number of decimal places for numeric values, and a wide variety of other things. User defined formats can represent values stored internally as codes as something more readable, such as 1=MALE and 2=FEMALE. Seeing a 1 or 2 on a report would not make as much sense as seeing MALE or FEMALE in most cases. For example

```
proc format;
value sex
1 = 'Male'
2 = 'Female'
;
run;
```

2.1.2. Specifying input/output format for non-SAS data

When data is read from non-SAS sources, such as a sequential file, the data step needs to know how to read files and put values into SAS variables. Data is read using the INPUT statement. The INPUT statement will often contain informats which will determine how SAS will interpret information on the file.

When writing to external files, a PUT statement will often contain formats which will produce output in a format suitable for the next processing step.

For example, to read 4 byte packed decimal data:

```
INPUT myvar PD4.;
```

Writing a 4 byte packed decimal value to the file is like this:

```
PUT myvar PD4.;
```

2.1.3. Grouping Observations

When summarizing data, it is often desirable to change how the data is aggregated for different types of analysis. This is easily done on the fly with SAS formats. A simple example is changing an aggregation from monthly to annual. All that needs to be done is change the format of the date variable used as a classification variable from YYMMDD4. to YYMMDD2. . Changing the format will change how the observations are grouped for analysis. Using user defined formats adds even more flexibility to the already extensive capabilities of the SAS System supplied formats.

The following example gets the number of batch jobs run in each period - each month in the first example and each year in the second.

Monthly Report:
 PROC FREQ DATA=batchjbs;
 TABLE rundate;
 FORMAT rundate yymmdd4.;

Annual Report:
 PROC FREQ DATA=batchjbs;
 TABLE rundate;
 FORMAT rundate yymmdd2.;

2.1.4. Table Lookups

While table lookups are often performed in SAS using a MERGE statement, using the PUT function with an appropriate format can perform the same function without sorting the main data file by the matching key. Also, fuzzy joins can be accomplished with formats because they handle fuzz factors and ranges of values. The data step with a MERGE statement does not allow this. PROC SQL does allow the same type of joining that a format allows in the data step - in fact, it is even more flexible and easier to use - but it is not as efficient for some situations and does not give the programmer the same level of control that a data step provides.

2.2. Syntax

The options for Proc format are as follows:

Option	Description
LIBRARY=	Name of the catalog which stores the formats. It is used when permanent formats are being created or reported on. The default is WORK.FORMATS. If a one level name is specified, then it is used as the libname and FORMATS is used as the catalog name.
FMTLIB	Prints information about the formats and informats.
PAGE	Causes FMTLIB output to have a separate page for each format catalog entry. Using this options causes FMTLIB to be automatically used.
MAXSELEN=	The number of characters used for the KEY (start and end values) when the format is written to a CNTLOUT data set or printed with the FMTLIB option. In general, this should be avoided because it can cause silly mistakes.
MAXLABLEN=	The number of characters used for the LABEL (result values) when the format is written to a CNTLOUT data set or printed with the FMTLIB option. In general, this should be avoided because it can cause silly mistakes.
CNTLIN=	Names the SAS data set being used as input to create formats and/or informats.
CNTLOUT=	Names the SAS data set containing information about the formats and informats in the SAS catalog. This can be used to generate custom reports providing much the same information that the FMTLIB option will generate.
NOREPLACE	Prevents existing formats from being overwritten by Proc Format. It can be a useful option for preventing mistakes, but it does not replace the need to use appropriate operating system security and library allocation.

NOTSORTED

Causes KEY entries to be searched linearly in the order they appear. This is set automatically if a format catalog is ported to another operating system.

2.3. Types of Formats

2.3.1. Value

These can be either numeric or character. All character format names must start with a "\$" as the first character. Numeric formats must not use a "\$" as the first character of the format name. Value formats are used to assign a specific new value based on the value that originally exists.

There are several options that can be specified on the Value statement, the most important is FUZZ=. FUZZ= is used for numeric formats only. For character formats, it is always 0. Setting FUZZ=0 will reduce space requirements for a numeric format, but your numeric data must then match exactly in order to get hit on the format. The default is FUZZ=1E-12 for numeric formats and 0 for character formats.

The FUZZ= option allows fuzzy matching, but you must be careful not to cause the lower and upper bounds of any of the KEY values to overlap by specifying too large a value for FUZZ. For example:

Good:

```
VALUE grades FUZZ=.4999
  1      =    'A'
  2      =    'B'
  3      =    'C'
```

Bad:

```
VALUE grades FUZZ=.5100
  1      =    'A'
  2      =    'B'
  3      =    'C'
```

Clearly, we have some overlap because 1.495 falls within the fuzz range of both 1 and 2, so we do not know if the grade should be an A or B. Having FUZZ=.5 would have caused the same problem because of 1.5 being in both ranges, but I used .5100 to avoid using a "boundary" condition for the example.

2.3.2. Picture

Picture formats are always numeric and are used to determine how numeric values should be displayed. This consists of one or more rules that are applied to the value to control the display. In addition, specific text values can be generated as with a Value format. These can be used for using characters like "CREDIT" and "DEBIT" instead of "+" and "-" for positive and negative numbers. It can also be used for including special characters as part of the value, forcing leading zeros, and a variety of other useful items.

2.4. Informats

Sometimes, it can be desirable to define your own informats instead of use standard SAS informats. Informats can be character or numeric. The input to an informat is always character and the result is determined by whether the informat is numeric or character. This is just the opposite of a value format. For example, if data for a variable called SEX comes on as MALE and FEMALE, you may want to store the values more compactly on your SAS data set as 1 and 2.

Using informats and formats in this manner will allow data to be stored more efficiently. Also, if the informat and format are permanently associated with the variable, then they will automatically be used as the default without special programming. They will also be self documenting in the sense that you can always find out how the informat and format work.

2.5. Limitations

Over the last several years, the limits on SAS user defined formats have increased dramatically. The LABEL, which is the value on the right side of the equal sign, can hold up to 200 characters. For character formats, the KEY, which is the value on the left side of the equal sign, can also contain up to 200 characters. These limits correspond to the maximum size of a SAS character variable.

The number of KEY values a format can have is now limited only by available memory. All formats being used in a data step or proc must reside in memory as the processes run. The limit used to be 32,760 in older releases of the SAS System. Obviously, some care must be used when building and using formats to insure that memory limitations will not be exceeded in your own operating system.

2.6. Features/Options

2.6.1. Formats referring to other formats

One really nifty new feature with Proc Format is the ability to have the result of a format be another format that is applied to the original value. This can provide a tremendous amount of flexibility when different ranges of values are to be displayed in substantially different ways.

The only restriction is that you must avoid recursive references - a format cannot call itself directly or indirectly.

Here is an example of displaying different expense amounts with different levels of precision.

```
PROC FORMAT;
VALUE expense
LOW - <0 = '<0'
0 - <100 = [dollar6.2]
100 - <1000000 = [dollar6.]
1000000 - HIGH = '>$1M';
```

User defined formats can also appear in the LABEL. This makes formats much more flexible for displaying output.

Using the data from this data step:

```
data temp;
amount = -22; output;
amount = 0; output;
amount = 34.55; output;
amount = 100.32; output;
amount = 98.95; output;
amount = 99.95; output;
amount = 99998.99; output;
amount = 99999.99; output;
amount = 299999.99; output;
```

We get this report:

OBS	AMOUNT
1	<\$0
2	\$0.00
3	\$34.55
4	\$100
5	\$98.95
6	\$99.95
7	\$99999
8	100000
9	>\$100K

2.6.2. NOTSORTED Option/Linear Search

Rather than do a binary search, it is possible to do a linear search of the KEY values in a format. This has two benefits. One is that items that will get the most frequent hits can be moved to the beginning of the list for faster processing. Informal tests run on MVS showed no improvement in processing time using a linear search over a binary search. For that reason, using the default binary search is probably better unless you have specific bench marking for your application that indicates otherwise. Another use of the NOTSORTED format option is that it allows formats to be moved between operating systems and still function without being recompiled. Any format transported between operating systems will use a linear search rather than a binary search because the sort sequence may be different on the new platform.

3. Infrastructure/Additional Topics

3.1. Picture Formats

Here are some examples of how picture formats can help in everyday processing.

3.1.1. Multiplication of Results

It can often be convenient to show ratios as percentages. This can be accomplished with a picture format.

```
DATA temp;
x = 0.395;

PROC FORMAT;
PICTURE pct (ROUND) OTHER = '099.9'
(MULT=1000);
```

```
PROC PRINT;
FORMAT x pct.;
```

The value of X displays as

39.5

3.1.2. Special Formatting

It may be desired to put special characters the variable's field when it is printed. This is illustrated by putting in a percent sign for the percentage.

```
DATA temp;
x = 0.395;

PROC FORMAT;
PICTURE pct (ROUND) OTHER='099.9%'
(MULT=1000);
```

```
PROC PRINT;
FORMAT x pct.;
```

The value of X displays as

39.5%

3.1.3. Dollars and Cents

Although SAS does have a DOLLAR. format, you may find that you need dollars and cents accuracy that is not available when using floating point numbers. You will find that since cents are fractions of a dollar, you may get slight errors which will cause your numbers to not add up. If you use the number of cents instead of the number of dollars, you will avoid this loss of accuracy because you will only be using integer values. In this case, the DOLLAR. format will not give you the correct result. You can use a picture format to have the value display the way you would expect.

```
DATA temp;
amt = 1054; * $10.54;
```

```
PROC FORMAT;
PICTURE cents (ROUND) OTHER=
'000,000,009.99' (MULT=1 PREFIX='$');
```

```
PROC PRINT;
FORMAT amt cents.;
```

The value of AMT displays as

\$10.54

3.2. Permanent Format Libraries

Permanent format libraries provide a way of storing formats so they can be used by many different SAS sessions without having to be redefined for each session. It provides for some processing efficiency in this manner. Also, it helps people to use standard formats that will deliver consistent results instead of creating their own format for every SAS session. Permanent formats are stored in a SAS catalog.

The LIBNAME of LIBRARY is special to the SAS System. Unless told otherwise by the FMTSEARCH option, if there is a LIBNAME of LIBRARY defined, SAS will look in its FORMATS catalog for formats and informats.

To Create a Permanent Format Library:

```
LIBNAME permlib 'system.file.name';
```

```
PROC FORMAT LIBRARY=permlib;
VALUE ...
INVALUE ...
PICTURE ...
```

3.2.1. SAS Options

In order to use permanent SAS formats, you must define a search path that allows your SAS session to see the formats and use them. The option that does this is the FMTSEARCH option. The library name or two level catalog name is used to identify each catalog. As many catalogs as desired can be concatenated together. If a format is define in two different catalogs, the format in the first catalog listed will be used. If only a library name is specified, then the catalog name of FORMATS is assumed.

If you are using permanent formats, you also might want to consider using the CBUFNO=1 option. This may speed the loading of your formats.

To Use Permanent Formats:

- * Your data set is called WHATEVER.;
- * The variable to print is X.;
- * The format for X is PERMFMT.;

```
LIBNAME permlib 'system.file.name';
```

```
OPTIONS FMTSEARCH=(permlib);
```

```
PROC PRINT data=whatever;
VAR x;
FORMAT x permfmt.;
```

Another Approach using the LIBRARY LIBNAME:

- * Your data set is called WHATEVER.;
- * The variable to print is X.;
- * The format for X is PERMFMT.;

```
LIBNAME library 'system.file.name';
```

```
PROC PRINT data=whatever;
VAR x;
FORMAT x permfmt.;
```

3.3. Data Sets - As Inputs and Outputs to Proc Format

Proc format will create an output data set from a format catalog; this is a CNTLOUT data set. It can also read a SAS data set to create formats; this is a CNTLIN data set. This capability better allows formats and data to be interchangeable which makes SAS formats an even more powerful tool for table lookups.

When creating a data set call FMT based on the CENTS format, we get one observation with the following values. In general, you can get more than one observation per format, but there is only one in this case. Also, you can select more than one format or choose to exclude certain formats.

```
PROC FORMAT CNTLOUT=fmt LIBRARY =
work;
SELECT cents;
```

Variable Name	Variable Value
FMTNAME:	CENTS
START:	**OTHER**
END:	**OTHER**
LABEL:	000,000,009.99
MIN:	1
MAX:	40
DEFAULT:	14
LENGTH:	14
FUZZ:	1E-12
PREFIX:	\$
MULT:	1
FILL:	-
NOEDIT:	0
TYPE:	P

SEXCL:	N
EEXCL:	N
HLO:	O

This data set, FMT, can now be used as a CNTLIN data set in a subsequent PROC FORMAT. This is a relatively painless way of moving formats across operating systems and getting them recompiled on the target machine.

```
PROC FORMAT CNTLIN=fmt LIBRARY =
WORK;
```

Notice that the SELECT statement is not used in this case. In fact, the way to control what is brought in from a CNTLIN data set is to use the WHERE data set option. A WHERE statement will not work. A SELECT statement will not prevent unlisted formats from compiling; it will only cause FMTLIB documentation to print out.

Here is how you would code to get only the CENTS format to compile. The LIBRARY option is unnecessary in this case because the WORK library is being used, but it is included to show how libraries must be specified in case you want to specify a permanent library..

```
PROC FORMAT CNTLIN = fmts (WHERE =
(fmtname = 'CENTS' )) LIBRARY=work;
```

4. Main Topics

4.1. Grouping Observations

One really nice feature of SAS formats is that provide a flexible means of grouping observations. Without changing any of the underlying data and by changing only the format used in a reporting procedure, the character of a report can be changed.

4.1.1. SAS Provided Formats

The formats included with the SAS System provide many useful tools for grouping data. Here are some examples of how to use them.

4.1.1.1. Dates

Date handling is generally a very strong feature in SAS. The wide selection of formats gives very flexible summarization capabilities. It would be worthwhile to review the new formats available in the latest releases. Check the appropriate technical reports containing documentation updates.

Here is an example:

The code for a Monthly report showing the number of orders would look like this:

```
PROC FREQ DATA=mydata;
TABLE ord_date;
FORMAT ord_date yymmdd4.;
```

To convert this to a Quarterly report, simply change the format:

```
PROC FREQ DATA=mydata;
TABLE ord_date;
FORMAT ord_date yyq.;
```

4.1.1.2. Beginning Characters

If you have a character variable such as 5 digit zip code, you might want to do some reporting based on 1 or 3 digit zip codes. Rather than create new variables, you can just use formats to change the way your data is grouped. Here is an example:

The code for a report showing the number of records in each 1 digit zip code would look like this:

```
PROC FREQ DATA=mydata;
TABLE zip_code;
FORMAT zip_code $char1.;
```

To convert this to a 3 digit zip code report, simply change the format:

```
PROC FREQ DATA=mydata;
TABLE zip_code;
FORMAT zip_code $char3.;
```

4.1.1.3. *Rounding Numbers*

Numbers used for classification purposes rather than as analysis variables can use formats to change how data is grouped. Here is an example. There are voter turn-out ratios for an election. There is data from each precinct. All of the values can range anywhere from 0 to 1 because they are ratios. If you want to get frequency counts showing how many precincts had a certain voter turnout, a format could determine the detail that you would see. You might want to summarize data based on 1 decimal place or 2. Here is how that would look with one decimal place:

```
PROC FREQ DATA=voting;
TABLE turn_out;
FORMAT turn_out 3.1;
```

If counts showing turn out to the second decimal place were desired, just change the format:

```
PROC FREQ DATA=voting;
TABLE turn_out;
FORMAT turn_out 4.2;
```

4.1.2. User Defined Formats

If the formats included with the SAS System do not meet your needs, you can create your own.

4.1.2.1. *Lists*

An example would be setting up a grade report to show how many people passed and how many were in other categories. Here is a format that will accomplish this for a data set containing grades.

```
PROC FORMAT;
VALUE $GRADE
'A','B','C' = 'Pass'
'D','F' = 'Retake Class'
OTHER = 'Withdrew';
```

4.1.2.2. *Ranges*

Continuous variables such as income can be used for classification purposes if you look at ranges rather than actual amounts for each person. For reporting from a data set containing a variable called INCOME, the following format could be used to aggregate data for purposes of analysis:

```
PROC FORMAT;
VALUE income
LOW - 0 = 'Negative'
0 - 10000 = '0-10K'
10000 - 30000 = '10-30K'
30000 - HIGH = 'Over 30K';
```

Naturally, the income ranges and number of income ranges can be easily changed to suit changing analytic needs.

4.2. Table Lookups

SAS formats are a very cost effective way of performing table lookups. Using formats can avoid a requirement to sort a large primary data set for a merge. Also, it may be possible to reduce the number of passes of the data by doing lookups with formats rather than a sort and merge for each lookup.

4.2.1. Basic Lookup

First we will look at typical lookups that involve equality conditions or ranges and have a character value as the result.

4.2.1.1. *Exact Match*

Here is an example of having a data set containing 3 letter code names for cities. There is a list of itineraries with each having a starting city and destination city stored as the variable START and END. A format contains the coordinates of each city and there is a formula for calculating the distance based on the coordinates.

```
DATA DISTANCE;
SET trips;
coord1 = PUT(start, coord.);
coord2 = PUT(end, coord.);
miles = distance(coord1,coord2);
```

This is rather straightforward and efficient compared to having to sort and merge the data twice in order to compute the miles traveled.

4.2.1.2. *Range*

Another case when a merge is not appropriate is when dealing with ranges of values. If certain demographic attributes were to be imputed based on some statistic such as the value of a household automobile, then ranges would probably be necessary. Here is an example:

```
PROC FORMAT;
VALUE carval
0 - 300 = 'Beater Mentality'
300 - 2000 = 'Low Cost Mentality'
2000 - 9000 = 'Used Car Acceptance'
9000 - 20000 = 'Likes New Car'
20000 - HIGH = 'What is a Used Car?';
```

```
DATA cartalk;
SET carinfo;
category = PUT(carvalue, carval.);
```

This assigns the category in one simple statement in the data step without having to do any messy IF-THEN-ELSE logic. In addition, if there are other attributes that need to be assigned to the person based on their category, they can be assigned using a format at the same time - all in only one pass of the data.

4.2.2.

When setting up a format as a lookup table, you will want to get the best performance in terms of both storage space and execution time. An example of a lookup table that could be set up as a format is a zip code table. For each zip code, there could be information concerning median income and median age of head of household. We might want to use this information to see how our customers' income and age compare with others in their area. This could be used as part of a marketing strategy or credit risk assessment. We would have to store zip codes as the KEY and median income and age as the label. Because SAS stores numbers as RB8., we would want to store our numbers that way to minimize conversion time.

The first step is to create a CNTLIN data set.

```
DATA fmt;
SET demograf (KEEP=zip medinc medage);
start = zip;
label = PUT(medinc,rb8.) !! PUT(medage,rb8.);
RETAIN ffmtname 'zipit' type 'C';
```

```
PROC FORMAT CNTLIN=fmt LIBRARY =
WORK;
```

```
DATA temp;
SET indata;
tempvar = PUT(zip, Szipit.);
medinc = INPUT(SUBSTR(tempvar,1,8),rb8.);
medage = INPUT(SUBSTR(tempvar,9,8),rb8.);
drop tempvar;
```

4.3. *Informats*

SAS uses informats to convert character strings into appropriate values for SAS variables. There are many informats provided by SAS and it is easy to write your own, if necessary.

4.3.1. SAS Provided

The SAS System comes with a wide variety of informats that will meet most of your needs. There are too many to go over in a short tutorial. Reviewing the SAS Language guide section on informats is highly recommended. There are many useful informats that you simply may not be aware of. Check the Technical Reports containing release updates too. New informats and formats are added to the new releases. Since they are new, they are a response to someone's recent need which may mirror your own.

4.3.2. User Defined

4.3.2.1. *Basic Example*

The following is an example of an informat which is used to read in data that may not be completely standardized. The objective is to convert anything starting with M to M and anything starting with either W or F to F. Anything else is a missing value. This will standardize entries such as MEN, MALE, WOMAN, and FEMALE.

```
proc format;
invalue $sex
'M' - < 'N' = 'M'
'W' - < 'X' = 'F'
'F' - < 'G' = 'F'
OTHER = ''
```

4.3.2.2. SAME_ key word

Here is an example of an informat using the SAME_ keyword. This is a convenient way of letting a range of values stay the same. This can also be done using an informat as the label of an informat. Notice that the numbers are treated as numeric values rather than character values on the left side of the equal sign.

```
proc format;
invalue rating(max=2)
0 - 4 = 1
5 - 20 = _same_
'A' - 'B' = 20
OTHER =
```

```
DATA rating;
INPUT raw $char2. @1 rating rating2. ;
DATALINES;
```

OBS	RAW	RATING
1	A	20
2	B	20
3	AA	20
4	BA	.
5	0	1
6	4	1
7	04	1
8	5	5
9	6	6
10	19	19

11	20	20
----	----	----

5. Conclusion

SAS Formats are versatile tools for displaying output, restructuring input, and performing table lookups. This is only a brief introduction to some of the ways formats can be used. You should look at various SAS manuals to see more examples and get a more detailed look at how formats and informats can benefit you.

6. References

SAS Language, Reference, Version 6, First Edition, Published by SAS Institute Inc.

SAS Procedures Guide, Version 6, Third Edition, Published by SAS Institute Inc.

SAS Technical Report P-222: Changes and Enhancements to Base SAS Software, Published by SAS Institute Inc.

SAS Technical Report P-242: SAS Software: Changes and Enhancements, Published by SAS Institute Inc.

Efficient Techniques for Random Access to SAS Datasets, SUGI 20 Proceedings, Published by SAS Institute

7. Author Contact Information:

If you have any questions or comments, please feel free to contact the author. Email is the preferred mode of communication.

Howard Levine

email: HowardLevine@fico.com

voice: 612-486-1793