SAS Formats: Uses and Abuses

Carry W. Croghan, US-EPA, Research Triangle Park, NC

ABSTRACT

SAS formats are a very powerful tool. They allow you to display the data in a more readable manner without modifying it. Formats can also be used to group data into categories for use in various procedures like PROC FREQ, PROC TTEST, and PROC MEANS (as a class variable). As with many powerful tools, there are some 'gotchas' to be aware of. Formats can crash a program without specific options. They can produce misleading information; this is especially true for picture format. There are many aspects of formats to consider, including 'built-in' or 'user defined' formats. A format may be temporary or permanent. The PUT and INPUT functions can be used to generate new variables using formats. If you have several format libraries, managing the formats can be difficult. The SAS dictionary tables are useful in managing the different formats. Code and output examples demonstrate the various uses and potential abuses of SAS formats

INTRODUCTION

This paper is an introduction to SAS formats and the related informats. Included is information on what a format is and why they are useful. There are examples of how to use and create formats. In addition, some potential errors are discussed.

WHY USE FORMATS?

Numeric values are efficient to store, especially integers. Inputting data using only the numeric key pad is quicker than alternating between number and letters. So data are usually collected and stored as numeric values. If you are very familiar with the data, understanding the coded values is not a problem. However, for those individuals that are not as closely involved with the data, the coded values can be incomprehensible. You could always hand out a code sheet with the data, but the reader may get a little annoyed at flipping back and forth from the data and the code sheet. SAS provided a better solution -- formats. Formats display the data in a more readable manner without modifying the data, making them a powerful tool.

SAS dates are an example of a type of data that is very difficult to interpret. SAS dates are integer values that represent the number of days since 01Jan1960. Although very efficient in storage, these values can be hard to interpret. SAS recognized this and provides a multitude of formats that are specific for dates, including: mmddyy10., date7., monyy5., and weekday1. Date=01Jan1960 is easier to understand than date=0.

ANATOMY OF A FORMAT

Let's look at the anatomy of a format. A name of a format has 4 parts: name, total width, dot, and number decimal locations. The name consists of 0 to 8 alphanumeric characters and cannot start or end with a numeric value. The width of the value is needed only if it can have multiple lengths. For example the 'datew.' format can have the widths 5, 7, or 9. The value date=0 formatted as 'date5.' is 01Jan, 'date7.' is 01Jan60, and 'date9.' is 01Jan1960. The dot is necessary for SAS to know that what you are referring to is a format and not a variable. The last part is the number of decimal places to be printed. For example, 'f8.2' is a numeric format with a width of 8 and 2 decimal places. It can accommodate values between -9999.99 to 99999.99. Note that both the minus sign and the decimal point take a space and must be accounted for in the width.

SAS has 3 kinds of formats: character, numeric, and datetime. A character format is used to write character data values that are determined by a character variable. A numeric format is used to write a numeric value determined by a numeric variable. A datetime format is used to write a value that represents the date, time, or datetimes stored in a numeric variable.

These are examples of SAS 'built-in' character formats:

Format	Description & example	
$CHARw.	Writes the character variable limiting the length to the specified width.	
	Text="abcdes";	abcd
	Format text $char4.;	
$HEXw.	Writes the character as hexadecimal	
	Text="a";	61
	Format text $HEX4.;	
$REVERSw.	Writes the character in reverse order and left aligns.	
	Text="abcdes";	edcba
	Format text $revers8.;	

These are examples of SAS 'built-in' datetime formats:

Format	Description & example	
DATEw.	Writes the date in the form of ddmonyy or ddmonyyyy	
	tDate= 15970;	22SEP03
	format tDate date7.;	
MMDDYYxw.	Writes the date in the form of mmddyy with specified separator.	
	tDate= 15970;	09:22:03
	format tDate mmddyyc8.;	
JULIANw.	Writes the date as the Julian date in the form yyddd or yyyyddd	
	tDate= 15970;	2003265
	format tDate julian7.;	
YYQxw.	Write the date as a year and quarter with the specified separator	
	tDate= 15970;	2003/3
	format tDate yyqs6.;	

These are examples of SAS 'built-in' numeric formats:

Format	Description & example	
BESTw.	SAS chooses the best notation for each numeric value	
	X= 29348 ;	29348
	X= 0.0423 ;	0.042
	X= 23 ;	23
	X= 2003 ;	2003
	format x best5.;	
COMMAw.d	Writes the numeric values with commas and decimal points	
	X= 29348 ;	29,348.0000
	X= 0.0423 ;	0.0423
	X= 23 ;	23.0000
	X= 2003 ;	2,003.0000
	format x comma13.4;	

Format	Description & example	
DOLLARw.d	Writes the numeric values with dollar signs, commas, and decimal points.	
	X= 29348 ;	$29,348.00
	X= 0.0423 ;	$0.04
	X= 23 ;	$23.00
	X= 2003 ;	$2,003.00
	format x dollar15.2;	
PERCENTw.d	Writes the numeric values as percentages (note it does both the multiplying by 100 as well as adding the % sign)	
	X= 29348 ;	2934800.0%
	X= 0.0423 ;	4.23000%
	X= 23 ;	2300.0000%
	X= 2003 ;	200300.00%
	format x percent12.5;	
ROMANw.	Writes the numeric values as Roman numerals. (note that fractions as well as 0 have no representation)	
	X= 29348 ;	*****
	X= 0.0423 ;	0
	X= 23 ;	XXIII
	X= 2003 ;	MMIII
	format x roman5.;	
WORDSw.	Writes the numeric values as words.	
	X= 29348 ;	twenty-nine thousand three hundred forty-eight
	X= 0.0423 ;	zero and four hundredths
	X= 23 ;	twenty-three
	X= 2003 ;	two thousand three
	format x words50.;	

SO WHAT IS AN INFORMAT

Informats are similar to formats but instead of specifying how to write data out they specify how to read data in. The application of informats is more limited than formats. Informats are used in input statements and functions. The structure of an informat is the same as for a format but the type specifies the output not the input. You cannot use informats and formats interchangeably but you can have formats and informats named the same. SAS has a 'built-in' format and informat called JULIAN. SAS recognizes the two things as different. However, don't assume that if there is a format and related informat exists, SAS will complain if it cannot find the format or informat it is looking for. Having informats and formats with the same name can be very confusing; I recommend that you keep the names different.

OTHER USES OF FORMATS

GROUPING

Besides making your data more readable, formats can be used to group data into categories for use in various procedures like PROC FREQ, PROC TTEST, and PROC MEANS (as a class variable). For example, you could get the mean sales values by quarter if you had the date of the sales. If your data look like this.

Obs	storeid	totsale	saledate
1	1	220	15717
2	1	257	15807
3	1	258	15534
4	1	295	15614
…			

then,

```
proc means data=sales maxdec=2;
     class saledate;
     var totsale;
     format saledate yyqn5.;
run;
```

produces the following output:

Analysis Variable : totsale

saledate	N Obs	N	Mean	Std Dev	Minimum	Maximum
20023	16	16	299.00	206.79	58.00	759.00
20024	12	12	353.67	252.81	123.00	895.00
20031	16	16	443.75	272.74	184.00	947.00
20032	16	16	426.75	261.74	107.00	907.00

The date data are still stored at the day level. The grouping has been accomplished by the use of the format.

Similarly, formats can be used to group the data in a PROC FREQ. Looking at the same dataset,

```
proc freq data=sales;
     table saledate;
     format saledate yyqn5.;
run;
```

produces the following output:

The FREQ Procedure

saledate	Frequency	Percent	Cumulative Frequency	Cumulative Percent
20023	16	26.67	16	26.67
20024	12	20.00	28	46.67
20031	16	26.67	44	73.33
20032	16	26.67	60	100.00

You can even group the data for a t-test using a format. To test for any significant change from 2002 to 2003, using a t-test, the code would be

```
proc ttest data=sales;
     class saledate;
     format saledate year4.;
     var totsale;
run;
```

which produces the following output:

The TTEST Procedure

Statistics

Variable	saledate	N	Lower CL Mean	Mean	Upper CL Mean	Lower CL StdDev
totsale	2002	28	235.24	322.43	409.61	177.77
totsale	2003	32	340.39	435.25	530.11	210.92
totsale	Diff(1-2)		-240.3	-112.8	14.62	208.27

…

```
                    T-Tests

Variable Method        Variances DF tValue Pr>|t|

totsale  Pooled        Equal     58 -1.77  0.0816
totsale  Satterthwaite Unequal   58 -1.79  0.0785
```

All these procedures used formatted data to combine data. The data were not modified. Also, there were no additional variables produced. Once you start generating your own formats, you can combine your data in a variety of ways. I usually create a format I call 'missing.'. Using my 'missing.' to format one of the key variables, I can test if those that do not answer that question are significantly different from the rest of the study population on the questions they do answer using a t-test.

If I want to create a categorical variable, I first format the variable into the categories. Then using PROC FREQ, I check that I have made good cut points on the distribution.
There are additional procedures for which formats can be used to group the data. Take a look at which procedures have class or by statements as good places to do this type of grouping using a format.

CREATING NEW VARIABLES

The PUT and INPUT functions can be used to generate new variables using formats. The PUT and INPUT functions should not be confused with the PUT and INPUT statements. The statements are methods of dealing with lines of data. The PUT statement is for outputting lines of data and the INPUT statement is for inputting lines of data. The PUT and INPUT functions are used to create new variables based on a value and a structure definition given by a format or informat.
The PUT function always results in a character variable. The anatomy is either varname = put(character_value, $character_format.) or varname=put(numeric_value, numeric_format.). The format and value must match data type. The value can be a constant, expression, or variable. For example, if a dataset has the variable 'id' coded as a numeric variable and you want to create a new id variable that is a character with leading zeros, the code would be:

```
new_id = put(id,z4.);.
```

Here is an example of how the data would look:

ID	NEW_ID
1	0001
10	0010
203	0203
397	0397

The type of variable produced by an INPUT function depends upon the type of the informat specified. The anatomy is varname = INPUT(value, $informat.) or varname = INPUT(value, informat.). For example, if the date information was stored in the dataset as a character variable in the form mm/dd/yy, to convert this to a SAS date is easy using the INPUT function and the informat mmddyy8: ndate = INPUT(cdate,mmddyy8.);. Since the informat mmddyy8. is a numeric informat, the type of ndate is numeric. The value for INPUT can also be a constant, expression, or variable.

SHOULD IT BE PERMANENT

You don't have to create a new variable to get a variable that is always displayed in a formatted manner. You can make a permanent format association for the variable. If a format is assigned in a data step or a procedure that modifies the data such as proc datasets or proc sql, then the association is permanent. (At least it is permanent until you change it either temporarily or permanently.) A format that is assigned in a procedure that does not modify the data is temporary. As soon as the procedure is completed the link between the variable and the format is gone.

The easiest way to find out what, if any, formats are assigned to a variable is by using a proc contents. If any formats are associated with your dataset, they will be listed.

You can also check a multitude of datasets at once by using the dictionary files. SASHELP.VCOLUMN has information on both format and informats that are assigned to a variable. Either print or select the libname and memname you are interested in.

HOW TO MAKE YOUR OWN FORMATS

Although SAS provides many 'built-in' formats, you will at some point want to make your own. This is done within the format procedure. There are three general types of formats you can make: value, invalue and picture. Value format is for decoding data. Invalue is for reading in coded values. Picture format defines a template for displaying numeric values.

VALUE

Value format is the type that I use most. It links a range, or set of values to a label. The data type, character or numeric, for the range must match the data type for the format. For example:

```
proc format;
     value sex       1 = 'Female'
                     2 = 'Male'
                     other='Error';
     value   $ ans   a = 'Always'
                     b = 'Sometimes'
                     c = 'Never';
     value agegrp    low-<13 = 'Pre-teen'
                     13-19   = 'Teenager'
                     20-30   = 'Young adult'
                     30-60   = 'Adult'
                     61-high = 'Senior';
```

INVALUE

If the variable comes in as a character and you want to translate it to numeric you can use an invalue format type. This creates an informat. Remember the data type is what the output is and not what the input is.

```
proc format;
     invalue   sex  'F' = 1
                    'M' = 2
                    other = .;
run;
```

Then you use the informat sex. as a parameter in an INPUT function (gender = INPUT(sex, sex.);), instead of using a series of if-then-else statements for the re-coding. For a variable with only two possible values, this is not much of a time savings. If you have a larger set of codes, the format already exists, or you have a dataset with the code and decodes in it, then this technique can save a lot of time and possible coding errors.

PICTURE

A picture format is a template for printing your numeric values. Like the value and invalue statements, you can specify ranges that you wish different templates to be applied. For example:

```
proc format;
picture
     low-<10000 = 'Less than 10k' (noedit)
     10000-high = '09.99M' (mult = 0.0001
     round);
run;
```

If the value is less than 10,000 then the comment 'Less than 10k' is printed. The noedit option specifies that the numbers in the label are to be printed as they are. For those values greater than or equal to 10,000, the value is multiplied by 0.0001 then is printed out. The zeros are place holders that are only used if the value is that large. Nines indicate places that either a zero or the value will be placed. Therefore,

If x is	This is what is printed
69767792.18	69.77M
35314.24	0.04M
904.48	Less than 10k
1579099.60	1.58M
-4683825.42	Less than 10k

Note that negative values also get the note of 'Less than 10K'. You will have to make a new range to accommodate the negative values if you want them treated differently.

OPTIONS

There are many different options you can use in a format procedure. The following options are common to INVALUE, VALUE, and PICTURE.

Options	Description	Default value
DEFAULT=length	Specifies the default length for the labels for the format.	Length of the longest label.
FUZZ=fuzz-factor	If a number does not exactly match the range, but is within fuzz-factor, then it is considered to be in the range.	1E-12 for numeric, 0 for character
MIN=length	Specifies the minimum length of a label.	1
MAX=length	Specifies the maximum length of a label.	40

Each type of format has its own set of options. Check the SAS procedure guide for details on the other options.

HOW TO DO MORE WITH FORMATS

In addition to the format level options, there are a many options that relate the format procedure.

HOW TO GENERATE A FORMAT FROM DATA

You can create a format from a dataset using the CNTLIN option. For example, if you have a dataset with the following structure

value (a character variable)	label
42101	NO2
42601	NO
42602	NOX
42603	CO

…

and you want to generate a format using the value as the range and label as the label. You first need to create a couple of additional variables that contain the type of the format and the name of the format (e.g., call it $chemnme). (Note that the same rules that apply to the name of a format also apply here.) The variable 'value' needs to be renamed to 'start'. Save this new dataset as chemname.

```
data chemname;
     retain fmtname "$chemnme"  type "c";
     set oldfile;
     rename value = start;
run;
```

The new dataset would look like

fmtname	type	start	label
$chemnme	c	42101	NO2
$chemnme	c	42601	NO
$chemnme	c	42602	NOX
$chemnme	c	42603	CO

…

Once you have a dataset with the necessary variables (named as SAS wants), then use

```
proc format cntlin= chemname;
run;
```

You have generated a format without having to do all the typing.

HOW TO GENERATE DATA FROM FORMAT

Generating data from a format is even easier. The CNTLOUT option creates a dataset with all the information for all the formats within a format catalog that you specify.

```
proc format cntlout=temp2;
run;
```

This produces a dataset called TEMP2 that has the following information:

Obs	FMTNAME	START	END	LABEL
1	AGEGRP	LOW	13	Pre-teen
2	AGEGRP	13	19	Teenager
3	AGEGRP	20	30	Young adult
4	AGEGRP	30	60	Adult
5	AGEGRP	61	HIGH	Senior
6	SEX	1	1	Female
7	SEX	2	2	Male
8	SEX	**OTHER**	**OTHER**	Error
9	ANS	a	a	Always
10	ANS	b	b	Sometimes

```
11    ANS        c           c           Never
12    SEX        F           F           1
13    SEX        M           M           2
14    SEX        **OTHER**   **OTHER**   .
```

Obs	MIN	MAX	DEFAULT	LENGTH	FUZZ	TYPE
1	1	40	11	11	1E-12	N
2	1	40	11	11	1E-12	N
3	1	40	11	11	1E-12	N
4	1	40	11	11	1E-12	N
5	1	40	11	11	1E-12	N
6	1	40	6	6	1E-12	N
7	1	40	6	6	1E-12	N
8	1	40	6	6	1E-12	N
9	1	40	9	9	0	C
10	1	40	9	9	0	C
11	1	40	9	9	0	C
12	1	40	1	1	0	I
13	1	40	1	1	0	I
14	1	40	1	1	0	I

Note that I have two FMTNAME = 'SEX' but one is an informat and the other is a format. You can tell which is which by looking at the variable TYPE.

The complete list of variables in a dataset generated by CNTLOUT is:

Variable	SAS label
DATATYPE	Date/time/datetime?
DECSEP	Decimal separator
DEFAULT	Default length
DIG3SEP	Three-digit separator
EEXCL	End exclusion
END	Ending value for format
FILL	Fill character
FMTNAME	Format name
FUZZ	Fuzz value
HLO	Additional information
LABEL	Format value label
LANGUAGE	Language for date strings
LENGTH	Format length
MAX	Maximum length
MIN	Minimum length
MULT	Multiplier
NOEDIT	Is picture string noedit?
PREFIX	Prefix characters
SEXCL	Start exclusion
START	Starting value for format
TYPE	Type of format

These are all the variables that you can specify in a CNTLIN dataset.

The EXCLUDE and SELECT statements can be used to limit or specify what formats are outputted to the dataset for the CNTLOUT output.

HOW TO STORE YOUR FORMATS

You don't want to have to re-create your formats over and over again. Also if several programmers are accessing the data, a central location for the formats not only saves time but limits errors as well. You would want to store your formats in a format catalog.

To create a format library, you use a special libname- LIBRARY.

```
libname library "C:\study formats";
```

Then in the proc format statement add the option library=LIBRARY. All the formats that you create within that procedure will be saved to a format catalog in the directory that you have specified.

As long as you include that libname statement in your later programs, SAS will find your user defined formats. SAS automatically searches WORK library as well as LIBRARY library for formats, in that order.

If you want to change the order of the search or have more than one directory that contains format catalog, use the option FMTSEARCH=, to specify the additional directories and search order.

For example, let's say there are four levels of formats. There are project and section level formats in addition to the study formats and those defined within the program stored on WORK. You want to search the program defined formats first, followed by the study, project and finally, section formats. `FMTSEARCH = (PROJECT, SECTION)` would search in that order since WORK and LIBRARY are not specified their order does not change. If, however, you want to search in the exact reverse order, then specify the `FMTSEARCH = (SECTION, PROJECT, LIBRARY, WORK)`.
If you have forgotten exactly which formats are in which libraries (which is easy to do if there are many libraries), you can use the fmtlib option in a proc format to output information on the formats and informats that are contained in a library. Then define the LIBRARY and then use the fmtlib option in PROC FORMAT.

```
libname LIBRARY "c:\a directory with a format
catalog in it.";

proc format lib=library fmtlib;
run;
```

produces a nice table for each format, including details on each level as well as details on the options for the format.

WARNINGS AND PROBLEMS

All powerful tools can be dangerous or just somewhat annoying - formats are no exception. You don't have to wear safety glasses but you need to always be aware of the pitfalls.
One annoying aspect of formats is that they can crash a perfectly good program. This occurs when SAS cannot not locate a specified format for the dataset. SAS gives you an error message and terminates the program. This usually happens when someone sends you data in which they have permanently associated some 'user-defined' formats and they haven't given the formats to you. (Also, when I forget to specify the correct library where I have stored my own user defined formats. But you won't do that). The option **NOFORMATERROR** will allow your program to run. This option just suppresses the error code that terminates the SAS program. It does not modify the data and the format associations. It is a stop gap solution. You should try to get or generate the formats that you are missing. If that is not possible, you can also unassign all formats within a data step with the code: format _all_;. However, you will lose all the information that the formats contained.

Another, and more insidious, problem with formats is that some times they give you misleading information. This happens when

you have not specified the correct format and the data do not fit into the structure that you have given. Picture formats are specifically dangerous. However, they are not the only culprits. The other day, I was reading in some data and needed to change the variable from character to numeric. I knew that the maximum size was 999.99 and I wanted to make sure I did limit the decimals to two places, so I used input(x,6.2). Looking at the data, I discovered that for those values that did not have the decimal values specified, I got incorrect data. For example, instead of getting 35.00 for 35, I got 0.35. So I used BEST6. and got what I was looking for. Always check the coded data against the decoded and make sure you have what you think you should have. If you are generating a new variable, compare the new to the old. If you are using a format to group data, make sure that the number in each group is what you were expecting.

If you are delivering a dataset to someone else, be sure there are no 'user-defined' formats associated with the dataset or else provide the format code to the data recipient. Format catalogs are hard to transfer. The code or a CNTLOUT dataset is a better way to send formats to someone else.

CONCLUSION

Hopefully you have learned something new about formats. I find them to be very useful tools. They can be both a time and space saver. They can make your data more readable. They can group your data for analysis. They can be used to generate new variables. The format procedure allows you to create 'user-defined' formats, generate dataset from formats or generate formats from datasets. You can store your formats in a format catalog. They cause problems if not properly managed. However, SAS provides many tools for handling formats, e.g., specifying search order, **NOFORMATERROR**, and FMTLIB. There are a lot of details that I could not go into here. Dig into the SAS manuals and do some additional reading on formats.

REFERENCES

Aster, Rick. Professional SAS Programmer's Pocket Reference. 3rd edition. Breakfast Communications Corp., Paoli, PA.

Cody, Ronald and Smith, Jeffrey. Applied Statistics and the SAS Programming Language. 4th Edition. Prentice Hall, Upper Saddle River, NJ.

SAS Procedures Guide, Version 8. SAS Publishing, Cary, NC.

SAS Language Reference: Concepts, Version 8. SAS Publishing, Cary, NC.

CONTACT INFORMATION

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

Author Name:	Carry W. Croghan
Company:	US-EPA
Address:	HEASD (MD E205-1)
City state ZIP:	RTP, NC 27709
Work Phone:	(919) 541-3184
Fax:	(919) 541-0239
Email:	croghan.carry@epa.gov

This paper has been reviewed in accordance with the United States Environmental Protection Agency's peer and administrative review policies and approved for presentation and publication.