PhUSE 2007

PaperTU06

Proc Format - Tricks and Traps

Christof Binder, BIOP Biometrical Practice, Basel, Switzerland

ABSTRACT

The Format procedure is one of the basic procedures that you can hardly avoid, even as a SAS beginner. However, there are a few subtle features and traps worth noting. This paper presents some of these features that come with formats. Care is taken to highlight traps along the way that might result in incorrect or unexpected results.

INTRODUCTION

The formats in SAS are often used to attach meaning to a coded variable or to change the way data is displayed in the output. In this paper some additional uses of formats are presented. Care is taken to point out the unexpected problems that may come up when using these additional ways to use formats.
The following sample dataset is used for all examples:

```
DATA smpdat;
    DO value = 1 TO 15;
        type = int((value / 10)) + 1; *type 1 and 2;

        IF ranuni(0) < 0.5 THEN cat = 'A'; *random 'A' or 'B';
        ELSE cat = 'B';

        IF (type eq 1) THEN DO;
            result = abs(sin(value)*5) + 1; *float value;
            resmiss = .;
        END;
        ELSE DO;
            result = int(sin(value)*10) + value; *integer value;
            resmiss = result;
        END;
        OUTPUT;
    END;

    valuefmt = vformat(value);
    PUT valuefmt=;
RUN;
```

LOADING DATASET WITH ASSOCIATED FORMATS THAT ARE NOT AVAILABLE

If a dataset has variables with formats associated that are not in the formats search path, the SAS system presents an error when trying to access these datasets (ERROR: Format <FORMATNAME> not found or couldn't be loaded for variable <VARIABLENAME>.). To be able to work with these datasets it is necessary to either get hold of these formats, or set the NOFMTERR system option to suppress these ERRORs. Either route makes it again possible to read and work with these data.
Alternatively, the formats can be deleted from the dataset by specifying an empty format statement:

```
DATA <datasetname>;
    SET <datasetname>;
    FORMAT _ALL_;
RUN;
```

TRAP

Don't trust the default numeric format BEST12.
If no format is specified for numeric variables, SAS uses BEST12. (http://v8doc.sas.com/sashtml/lgref/z0184495.htm),

as confirmed by the PUT statement in the data step, above. SAS Online Documentation states that:

"Formats never change or truncate the internally stored data values."
(http://v8doc.sas.com/sashtml/lrcon/z0919961.htm)

But the documentation fails to highlight that SAS, operating in a binary world, is sometime unable to store internally the correct number. The default format BEST12. can hide this discrepancy (round-off error) between the true decimal result and SAS' internally stored number. Consider the following example taken directly from SAS' technical support document TS-654 (http://support.sas.com/techsup/technote/ts654.html, and the more detailed http://support.sas.com/techsup/technote/ts230.html):

```
DATA _null_;
  x = 15.7-11.9;
  IF x=3.8 THEN PUT 'equal';
  ELSE PUT 'not equal';
RUN;
```

This data step PUTs 'not equal' to the log, even though any decimal novice knows that 15.7-11.9 = 3.8. Again, the problem is that SAS & computers are not even decimal novices, let along decimal experts.

The default numeric format, BEST12., greatly interferes with programmers' ability to discover and understand problems introduced by this shortcoming of computer storage. If you try to inspect the data set "a" created above, you will see the correct result:

VIEWTABLE: Work.A

	x (default best12. format)
1	3.8

So what is wrong? Simply applying, instead, the widest possible BESTw. format, BEST32. reveals how SAS inexplicably introduced rounding error into this simple decimal calculation:

VIEWTABLE: Work.A

	x (non-default BEST32. format)
1	3.79999999999999

Rounding algebraic results to the appropriate level of precision, after even the simplest computations, is essential when working with SAS, and fully the responsibility of the programmer. SAS, unfortunately is unable to automatically correct its rounding errors. Imagine the potential for mistake and confusion if the above calculation and result contributed, for example, to an analysis of abnormal lab results that flagged transformed values >= 3.8.

Recommendations: Never accept SAS default numeric format of BEST12.; instead, explicitly apply BEST32. format to all numeric variables that do not have a more precise requirement. And ALWAYS round results after a chain of algebraic calculations. Keep both rules in mind when working with intermediate, temporary data sets.

OUTPUT IN A SPECIFIC ORDER

The order in which the groups are printed can be influenced by a format that is associated with a variable. Depending on whether you want your output to be sorted by the raw or the formatted values the procedure can be instructed to use the format for sorting or not. [Alternative: Programmers can choose to sort by the raw or the formatted values, as appropriate.

EXAMPLE

```
PROC FORMAT;
    VALUE ftype  1 = 'Group B'
                 2 = 'Group A'
                 ;
RUN;

PROC FREQ DATA=smpdat ORDER=formatted;
    table type;
    FORMAT type ftype.;
RUN;
```

```
PROC FREQ DATA=smpdat /* ORDER=internal */;
    table type;
    FORMAT type ftype.;
RUN;
```

GROUPING INTO CLASSES

It is often desirable to have the output grouped into classes, rather than having one line of output per value present in the data. Creating in format that contains these classes and using it in the output can achieve this goal.

EXAMPLE

```
PROC FORMAT;
    VALUE fclass
        low-2      = 'very small'
            2-5    = 'reasonable'
            5-high = 'big';
RUN;

PROC FREQ DATA=smpdat;
    table result;
    FORMAT result fclass. ;
RUN;
```

TRAP

Again, the internal representation of floating point numbers (representation error) and the imprecise storage of floating point results (rounding error) can give some unexpected results. Either use the FUZZ option of PROC FORMAT to catch these border cases or explicitly round floating point values. If not done carefully it can make it very hard to find problems in your output, as demonstrated above.

MULTILABLE FORMATS

Some procedures accept multilable formats, i.e. formats that assign more than one label to a value. This can be useful if a specific value can be both “high” and “interesting”.

EXAMPLE

```
PROC FORMAT;
    VALUE multi (multilabel)
        0-5   = 'low (0-5)'
        6-12  = 'med (6-12)'
        13-20 = 'high (13-20)'
        11-13 = 'flag (11-13)'
        ;
RUN;

PROC TABULATE DATA=smpdat ORDER=data;
    CLASS value / mlf;
    TABLE value  all,n='count' ;
    FORMAT value multi.;
RUN;
```

TRAP

If a procedure does not accept multilable formats (like PROC PRINT) it has to select a sub-format. The sub-format selected is the first one based on the sorted ranges provided and not the first one provided! In the following example the output for the values 13 is “flag” and not “high”, however for 11 and 12 it is “med”.

```
PROC PRINT data = smpdat(where=(value gt 9));
    VAR value;
    FORMAT value multi.;
RUN;
```

Output:

```
Obs       value

10    med (6-12)
11    med (6-12)
12    med (6-12)
13    flag (11-13)
14    high (13-20)
15    high (13-20)
```

If only 'low', 'med' and 'high' are to be displayed a second – non multilable - format without the 'flag' category is needed.

OUTPUT OF MISSING CATEGORIES

The PROC FREQ procedure produces, by default, only output for categories that are present in the data. If output for missing classes is also requested this can be done by using the option PRELOADFMT combined with the PRINTMISS option.

EXAMPLE

```
PROC TABULATE DATA=smpdat;
    CLASS type /PRELOADFMT;
    VAR resmiss;
    TABLE type*resmiss /PRINTMISS;
    FORMAT type fclass.;
RUN;
```

USE DIFFERENT FORMATS FOR THE SAME VARIABLE

Sometimes it is necessary to output one variable with different formats depending on the contents of a second variable. This is especially true for normalized datasets where several measurements are stored in a result variable and the type of measurement in a second. What is needed are dynamic formats that allow this flexibility. The procedures PUTN and PUTC allow for this flexibility:

EXAMPLE

```
PROC FORMAT;
    VALUE ftyp 1 = '5.1'
               2 = 'z5.';
RUN;

DATA out_;
    SET smpdat;
    LENGTH formatted $8;

    typef = PUT(type, ftyp.);
    formatted = PUTN(result, typef);
RUN;
```

TRAP

Even though the PUTN does look like a common PUT statement, don't put a period after the format name. The two lines can be combined to give a more compact syntax and make it less likely to trigger the urge to put a period in the wrong place:

```
    formatted = putn(result, put(type, ftyp.));
```

USE FORMATS TO COMPLETE ARRAY SUMMARIES

A technique similar to the previous one is useful for categorizing and counting data using arrays, even arrays with dynamic dimensions. As a simple example, use the following formats to count separate summaries of categories "A" and "B" in the sample data set:

```
PROC FORMAT;
    VALUE $abcat  'A' = 1
                  'B' = 2;
```

```
    VALUE fcat    low-2  = 1
                  2-5    = 2
                  5-high = 3;
RUN;


DATA _null_;
    SET smpdat end=NoMore;

    ARRAY ab[2,3] _temporary_;
    ab[ putc(cat,'abcat'), putn(result,'fcat') ] + 1 ;

    /* take a look at the results of these 2 lines of code */
    IF NoMore THEN DO;
        DO idx = 1 TO dim1(ab);
            DO jdx = 1 TO dim2(ab);
                PUT ab[idx,jdx]= @;
            END;
        PUT;
        END;
    END;
RUN;
```

TRAP

Using this technique too freely can result in code that is difficult to read, and possibly hard to maintain by programmers unfamiliar with arrays and formats.

CONCLUSION

This paper has shown that formats in SAS are more than just a way to add meaning to a coded variable or an easy way to present data in a consistent way. With some creativity and cleverness results can be reached with quite compact code. But as with all clever code there are some traps along the way that are worth pointing out, because once caught in them it may take a few hours of debugging time to free yourself.

REFERENCES

The SAS online documentation (http://v8doc.sas.com/sashtml).

ACKNOWLEDGMENTS

The author would like to thank Dante diTommaso for his much valued input.

CONTACT INFORMATION

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

Christof Binder
Biometrical Practice, BIOP
Centralbahnstrasse 9
CH-4051 Basel, Switzerland
Email: christof.binder@biop.ch
Web: www.biop.ch