PhUSE 2015

Paper CC03

Fun with Formats

Sarah Berenbrinck, Independent, UK

ABSTRACT

The paper will show some ways of using formats to support programming, covering the basics of using formats to change the appearance of values and using formats to modify values. It will include how to align the output when using put statements, ways to suppress errors if you are expecting dirty data and still want your program to run. Format catalogs can be converted into datasets and datasets to formats, including how this is done and some of the uses. Creating and using picture formats to produce custom formats and fully customize outputs.

INTRODUCTION

This paper will show some uses for SAS ® formats and some example code.

BASICS

Here is some code to create a dataset (using datalines) which will be used to demonstrate some formats.

```
data phuse1 ;
 length date $9 ;
  input date $ ;
  datalines  ;
  13OCT2013
  12OCT2014
  11OCT2015
  ;
run ;
```

This dataset has 1 column and 3 rows, the data is dates, these dates are fine for listing however if they are needed in a different style or required for calculations then some manipulation will be required.

PUT AND INPUT STATEMENTS

PUT and INPUT statements can be used to create new variables using formats, the following datastep uses formats that are supplied with Base SAS software.

```
data phuse2 ;
  set phuse1 ;
  datenum=input(date,date9.) ;
  dateymd=put(datenum,yymmdd10.) ;
  year=put(datenum,year.) ;
  year1=put(year,$4.) ;
  year2=input(year,3.) ;
  year3=input(year,best.) ;
run ;
```

INPUT statements create numeric variables and PUT statements create character variables.

The format used in each line is the red text, format names always contain a. (usually at the end). Also character formats are started with a $ symbol.

This datastep creates 6 new variables using PUT and INPUT statements.

VIEWTABLE: Work.Phuse2

	DATE	DATENUM	DATEYMD	YEAR	YEAR1	YEAR2	YEAR3
1	13OCT2013	19644	2013-10-13	2013	2013	201	2013
2	12OCT2014	20008	2014-10-12	2014	2014	201	2014
3	11OCT2015	20372	2015-10-11	2015	2015	201	2015

DATE is the variable that was created by datalines.

DATENUM is the dates from DATE as SAS numeric values, this statement uses date9. which is the format that the original dates were in. These numbers can be used in calculations such as days since event or durations.

DATEYMD contains character dates shown in a different way (year, then month, then day using the number identifiers).

YEAR is created using the format year. this takes the year part of a date, as it has been used with PUT, it creates a character variable.
The final 3 variables are created from the 4 character variable YEAR.
YEAR1 uses PUT and a character format to create a 4 character variable.
YEAR2 uses INPUT and 3. this creates a numeric variable, however the 3 means it only considers 3 characters so the value is truncated.
YEAR3 uses best. this means that SAS considers the best format for that number and will not lead to truncation. best. is good to use if you are unsure of the content however it will not align data for reporting so can lead to messy looking output.

BEST.

An example of using best. or a defined number.

```
data best ;
do i=1 to 6 ;
  x=ranuni(1) ;
  y=put(x,best.) ;
  z=put(x,4.2) ;
  output ;
end ;
run ;
```

VIEWTABLE: Work.Best

	I	X	Y	Z
1	1	0.1849625698	0.1849625698	0.18
2	2	0.9700887157	0.9700887157	0.97
3	3	0.3998243061	0.3998243061	0.40
4	4	0.2593986454	0.2593986454	0.26
5	5	0.9216025779	0.9216025779	0.92
6	6	0.9692773498	0.9692773498	0.97

best. gives the full number, 4.2 gives 2 decimal places which may be all that is relevant, note that for 4.2 the .2 gives the number of decimals while the 4 gives the full length of the string so in this case only allows for 1 character before the decimal place.

CREATING A FORMAT WITH PROC FORMAT

Value formats can be created with the value statement followed by the name of the format. Character formats, to be used on character values have names starting with a $.

```
proc format ;
  value phusel 2013 = 'Brussels'
               2014 = 'London'
               2015 = 'Vienna'
        ;
  value $phusec 'Brussels' = 'Belguim'
                'London' = 'England'
                'Vienna' = 'Austria'
        ;
run ;
```

Apply phusel. format to the year values from above and then apply $phusec. to those cities.

```
data phuse3 ;
  set phuse2 ;
  location=put(year3,phusel.) ;
  country=put(location,$phusec.) ;
run ;
```

YEAR3	LOCATION	COUNTRY
2013	Brussels	Belguim
2014	London	England
2015	Vienna	Austria

CHANGING APPEARANCE WITH FORMATS

Format and attrib statements can be used to apply formats to variables; these change the appearance of the variables while SAS maintains the original values.

```
data phuse4(keep=year3 yearx yeary location) ;
  format yearx phusel. ;
  attrib yeary format=phusel. ;
  set phuse2 ;
  yearx=year3 ;
  yeary=year3 ;
  location=put(year3,phusel.) ;
run ;
```

VIEWTABLE: Work.Phuse4

	YEARX	YEARY	YEAR3	LOCATION
1	Brussels	Brussels	2013	Brussels
2	London	London	2014	London
3	Vienna	Vienna	2015	Vienna

YEARX, YEARY and LOCATION all appear the same however YEARX and YEARY are the years with the fomat applied, while LOCATION actually contains those values.

ALIGNMENT

When reporting data changing the alignment can improve the output, one way to do this is in the PUT or INPUT statement by using -l, -c or -r before the), for left, centre or right aligned.

```
data phuse5 ;
  set phuse2 ;
  locl=put(year3,phusel.-l) ;
  locc=put(year3,phusel.-c) ;
  locr=put(year3,phusel.-r) ;
run ;

proc print data=phuse5 ;
  var year3 locl locc locr ;
run ;
```

Obs	YEAR3	LOCL	LOCC	LOCR
1	2013	Brussels	Brussels	Brussels
2	2014	London	London	London
3	2015	Vienna	Vienna	Vienna

SUPPRESSING ERRORS

Sometimes programming is done on dirty data.

```
data phuse6 ;
 length date $9 ;
  input date $ ;
  datalines  ;
  31JAN2016
  31FEB2016
  31MAR2016
  ;
run ;
```

The data may be corrected before it is classed as final, however until this happens it is useful if programs run so code can be prepared and debugged.

This dataset contains 3 dates and trying to format them as done above causes issues for 31FEB2016 as this date does not exist.

```
data phuse7 ;
  set phuse6 ;
  datenum=input(date,date9.) ;
  dateymd=put(datenum,yymmdd10.) ;
  check=_error_ ;
run ;
```

```
NOTE: Invalid argument to function INPUT at line 311 column 11.
DATE=31FEB2016 DATENUM=. DATEYMD=. CHECK=1 _ERROR_=1 _N_=2
NOTE: Mathematical operations could not be performed at the following places.
      missing values.
      Each place is given by: (Number of times) at (Line):(Column).
      1 at 311:11
NOTE: There were 3 observations read from the data set WORK.PHUSE6.
NOTE: The data set WORK.PHUSE7 has 3 observations and 4 variables.
NOTE: DATA statement used:
      real time           0.00 seconds
      cpu time            0.00 seconds
```

There is a note in the log and the internal variable _error_ has been set to 1, this does not stop the program running but does make the log look messy.

Adding a ? before the format suppresses the notes in the log.

```
data phuse8 ;
  set phuse6 ;
  datenum=input(date,?date9.) ;
  dateymd=put(datenum,?yymmdd10.) ;
  check=_error_ ;
run ;
```

```
DATE=31FEB2016 DATENUM=. DATEYMD=. CHECK=1 _ERROR_=1 _N_=2
NOTE: There were 3 observations read from the data set WORK.PHUSE6.
NOTE: The data set WORK.PHUSE8 has 3 observations and 4 variables.
NOTE: DATA statement used:
      real time           0.00 seconds
      cpu time            0.00 seconds
```

The internal SAS variable _error_ is still set to 1 and SAS will only allow a set number of these before it gives an error and stops the program running. If you require clean data you may want the program to error and alert you to the issue, however you may wish the program to run while you develop the code and wait for clean data.

Adding a ?? before the format suppresses the note in the log and stops _error_ from being flagged.

```
data phuse9 ;
  set phuse6 ;
  datenum=input(date,??date9.) ;
  dateymd=put(datenum,??yymmdd10.) ;
  check=_error_ ;
run ;
```

```
NOTE: There were 3 observations read from the data set WORK.PHUSE6.
NOTE: The data set WORK.PHUSE9 has 3 observations and 4 variables.
NOTE: DATA statement used:
      real time           0.00 seconds
      cpu time            0.00 seconds
```

The other errors that may occur, are from using formats that don't exist or are not present in your formats catalog.

```
data phuse10 ;
  set phuse1 ;
  datenum=input(date,date9.) ;
  dateymd=put(datenum,test.) ;
run ;
```

```
330  data phuse10 ;
331    set phuse1 ;
332    datenum=input(date,date9.) ;
NOTE: SCL source line.
333    dateymd=put(datenum,test.) ;
                           -----
                           48
ERROR 48-59: The format TEST was not found or could not be loaded.

334  run ;

NOTE: The SAS System stopped processing this step because of errors.
WARNING: The data set WORK.PHUSE10 may be incomplete.  When this step was stopped
NOTE: DATA statement used:
      real time           0.01 seconds
      cpu time            0.01 seconds
```

The format test. does not exist so the program errors, this may happen if you have inherited code from a different project and you may wish to still run the program.

```
options nofmterr ;
```

This options suppresses errors from formats, the log will now display:

```
337  data phuse10 ;
338    set phuse1 ;
339    datenum=input(date,date9.) ;
NOTE: SCL source line.
340    dateymd=put(datenum,test.) ;
                           -----
                           484
NOTE 484-185: Format TEST was not found or could not be loaded.

341  run ;

NOTE: There were 3 observations read from the data set WORK.PHUSE1.
NOTE: The data set WORK.PHUSE10 has 3 observations and 3 variables.
NOTE: DATA statement used:
      real time             0.03 seconds
      cpu time              0.01 seconds
```

It highlights the nonexistent format but still runs the program.

When you have finished testing code you may wish errors from formats to show in the log, this can be switched back on using:

```
options fmterr ;
```

The option nofmterr also suppresses errors and allows datasets to open if you have data that has been formatted with formats that are not in your formats catalog, however you may lose information only contained in the formats, e.g. the dataset may have treatment as A and B with a format to assign it to investigational product and comparator.

CATALOGS TO DATASETS

The formats created earlier exist in a formats catalog

Contents of 'Work.Formats'

Name	Size	Type
Phusel	0.3KB	Format
Phusec	0.2KB	Formatc

However from here they cannot be opened for viewing, if you which to view the values you have to either print the format or create a dataset.

Proc format can be used to write the values to a dataset.

```
proc format library=work cntlout=phusefmt ;
run ;
```

This creates dataset containing the values and labels for all user generated formats, these may have been created using proc format as above or they could be from an imported formats catalog.

VIEWTABLE: Work.Phusefmt

	FMTNAME	START	END	LABEL
1	PHUSEL	2013	2013	Brussels
2	PHUSEL	2014	2014	London
3	PHUSEL	2015	2015	Vienna
4	PHUSEC	Brussels	Brussels	Belguim
5	PHUSEC	London	London	England
6	PHUSEC	Vienna	Vienna	Austria

It is also possible to create formats from datasets. The dataset must contain the variables for format name, start value, end value and label, (extra variables would not cause issues, in the example below only the variables needed are kept to make the output smaller).

```
data phuse11(keep=fmtname start end label) ;
  set phuse3 ;
  fmtname='phusell' ;
  start=year3 ;
  end=year3 ;
  label=location ;
run ;
```

VIEWTABLE: Work.Phuse11

	FMTNAME	START	END	LABEL
1	phusell	2013	2013	Brussels
2	phusell	2014	2014	London
3	phusell	2015	2015	Vienna

This dataset can now be used to create a format.

```
proc format library=work cntlin=phuse11 ;
run ;
```

```
358  proc format library=work cntlin=phuse11 ;
NOTE: Format PHUSELL has been output.
```

This fomat can be viewed using cntrlout.

```
proc format library=work cntlout= phusefmt2 ;
run ;
```

VIEWTABLE: Work.Phusefmt2

	FMTNAME	START	END	LABEL
1	PHUSEL	2013	2013	Brussels
2	PHUSEL	2014	2014	London
3	PHUSEL	2015	2015	Vienna
4	PHUSELL	2013	2013	Brussels
5	PHUSELL	2014	2014	London
6	PHUSELL	2015	2015	Vienna
7	PHUSEC	Brussels	Brussels	Belguim
8	PHUSEC	London	London	England
9	PHUSEC	Vienna	Vienna	Austria

The new format phusell is in the middle.

PICTURE FORMATS

Picture formats can be used to add further customization to the output generated by formatting variables.

```
data phuse12 ;
  input amount ;
  datalines  ;
  5
  5.50
  10
  30
  1000
  2000
  ;
run ;
```

This creates a dataset containing a list of numbers, imagine them to be prices in Euros, you may want to display them with the Euro symbol. We will create some formats to view these amounts in different ways.

```
proc format library=work ;
  picture euros low-high='000,000€' ;
  picture eurosr (round) low-high='000,000€' ;
  picture pound low-high='000,000' (mult=0.73 prefix='£') ;
  picture poundr (round) low-high='000,000' (mult=0.73 prefix='£') ;
run ;
```

Apply these formats to our list of amounts.

```
data phuse13 ;
  set phuse12 ;
  euro=put(amount,euros.) ;
  euror=put(amount,eurosr.) ;
  pound=put(amount,pound.) ;
  poundr=put(amount,poundr.) ;
run ;
```

VIEWTABLE: Work.Phuse13

	AMOUNT	EURO	EUROR	POUND	POUNDR
1	5	5€	5€	£3	£4
2	5.5	5€	6€	£4	£4
3	10	10€	10€	£7	£7
4	30	30€	30€	£21	£22
5	1000	1,000€	1,000€	£730	£730
6	2000	2,000€	2,000€	£1,460	£1,460

The first format euros. adds the symbol for euro, however the format only displays integers so for 5.5 it only takes the integer part.
The second format euror. rounds the value to the nearest integer and adds the symbol.
Another thing that can be done with picture formats is to add a multiplier, the examples above convert from Euros to pounds sterling.
These 4 picture formats treat all values the same way, it is possible to create formats which apply a different format depending on the value input.

```
proc format library=work ;
  picture pounde (round) low-100 ='000,000' (mult=0.73 prefix='£')
                         101-high='Too expensive'
                ;
run ;
```

This format treats values of 100 and below one way and vales of 101 and above a different way.

```
data phuse14 ;
  set phuse12 ;
  poundr=put(amount,poundr.) ;
  pounde=put(amount,pounde.) ;
run ;
```

VIEWTABLE: Work.Phuse14

	AMOUNT	POUNDR	POUNDE
1	5	£4	£4
2	5.5	£4	£4
3	10	£7	£7
4	30	£22	£22
5	1000	£730	Too expensive
6	2000	£1,460	Too expensive

CONCLUSION

Formats are one way to transform SAS data and output to match your standards or better express what you are trying to show.

REFERENCES

http://support.sas.com/onlinedoc/913/docMainpage.jsp
http://support.sas.com/onlinedoc/913/getDoc/en/lrdict.hlp/a000309859.htm

CONTACT INFORMATION

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

Sarah Berenbrinck
Independent
37 Westmill Road
Hitchin
Email: sberenbrinck@gmail.com

Brand and product names are trademarks of their respective companies.