

Make Your Life Easier with SAS Formats

Sean Burnett
Burnett Consulting Corp.
Victoria, B.C

1. Overview

This paper demonstrates how to use SAS Formats to simplify adhoc programming. By keeping an up to date, well document format library, you can make your SAS programming life a lot easier. This paper will give you a step by step guide to create and store your own SAS Format library. It will cover the following topics:

- Common uses of formats;
- The difference between Picture Formats, Value Formats, and Informats;
- Using Proc Format to build your own Value Formats as an alternative to lookup tables;
- A step by step example of creating character and numeric formats;
- When and when not to use formats ;

2. Common Uses of SAS Formats

You have likely seen or used SAS formats in the following manner;

```
Data temp;  
  Set invoices;  
  Datetext=put(invdate, date7.); /* makes a SAS date appear as eg "23JUN67"*/  
  Money=put(invcost, dollar11.2); /* makes a numeric appear as eg."$123,123.95" */
```

Run;

Or...

```
Data invoices;  
  Set invoices;  
  Format invdate date7.  
          Invcost dollar12.2
```

run;

In both instances, it is a way of reformatting a number or coded value to appear as more readable text. It is important to note that "datetext" and "money" are both character variables. Formats always return character variables. To understand this, consider the nature of formats – they format numbers into more readable strings, such as 123123.95 to \$123,123.95 or, 1674 to "23JUN67" – clearly more readable by humans, but not as efficient for computers. Also note that formats are a type of "mask". The conversion could be a simple comma for the thousands digit or a complicated conversion such as "23JUN1967" from the sas date 2370.

There are essentially two different kinds of formats;

- **Picture formats:** to change, for example, 123123123.1 to \$123,123,123.10 and;
- **Value Formats:** look up a value and return a text string e.g. A8822 to "VULCANIZED X-RADIAL 120"

2.1. The Difference Between Formats and Informats

Informats are similar to formats in many ways, yet opposite. A format converts a value to (in general) a more readable string, and informats are often used to read text in order to store numeric values into SAS datasets. The difference between formats and informats is the same difference as exists between PUT and INPUT.

For example, SAS holds the date "23JUN1967" as the value 2730. For printing a report, you would want SAS to convert the date 2730 to "23JUN1967" or,

```
Datetext=Put(2730, date9.);
```

Likewise, you may wish to input a date in a convenient format, but store it as a SAS date so that it may be manipulated,

```
Sasdata=input('23JUN1967', date9.);
```

In this case, SAS happens to call an informat by the same name as the format. This is just by convention.

A PUT statement always returns a character variable.

3. Using Proc Format

There are several ways to use proc format and formats in general. This paper will not attempt to cover them all, but will try to cover the common and useful ones. Since Value Formats have the most bang per buck, our concentration will focus here. But first, a brief description of what we are glossing over: Picture Formats and user-defined Informats.

Picture formats are a way of performing a character manipulation on a numeric (as above, 123123123.1 to \$123,123,123.10). Since SAS already provides almost every imaginable picture format to do this type of transformation, we will not go into making our own.

User-defined Informats are another use for Proc Format. By creating a custom informat, you may read in data that contains Y/N and store it to be numeric 1/0. Essentially, once you learn how to create your own format, it is a small task to extend that into building your own informats. Again, SAS already provides many commonly used informats to read in odd data.

Value formats perform the function of a lookup table. Often, there is the need to link codes such as "1", "2", and "3" to meaningful names like "Truck", "Car" and "Motorcycle". Generally, this task can be done without a Proc Format by performing a dataset merge, or "if ... then... else" statements. A basic Value Format can be coded as follows:

```
Proc format;  
  Value agegrp  
    Low-<13='Child'  
    13-<20 ='Teenager'  
    20-high='Adult'  
;  
run;
```

This creates a numeric format – one that maps a number to a text string. By running this code in a SAS session, you have created the format "agegrp." which is stored in your temporary WORK library. Once you have run the code, you may use the format "agegrp." in the following way...

```
Data temp;  
  Set ages;  
  Agegroup=put(age, agegrp.); /* agegroup will now read "Child", "Adult"...*/  
Run;
```

You can also embed functions...

```
Data temp;  
  Set ages;  
  Agegroup=put(((dtofserv-birthdt)/365.25), agegrp.);  
Run;
```

Or, you can bypass creating any variables and use the format directly in a proc print, proc tabulate, etc.:

```
Proc print data=ages;  
  Format age agegrp.;  
Run;
```

Another variation on the Proc Format is to map character strings to other character strings ("character format")

```
Proc format;  
  Value clients  
    'A02'='Fred'  
    'A03'='Charlene'  
    'A04'='Edwin'  
    'A05'='Mr. T'  
;  
run;
```

For both number and character formats, typing in these values could become labour intensive if the number of names you have is large. However, if you store your data in a table, you can turn it into a format easily by using the CNTLIN option in Proc Format.

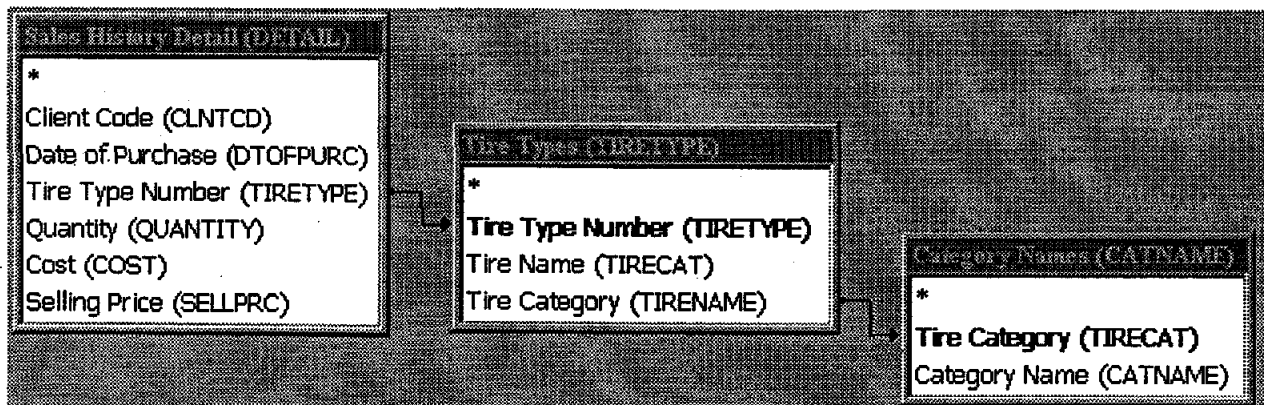
You can use Proc Format to convert your lookup tables directly into formats with a little manipulation. Specifically, your dataset must have the index variable renamed to be called "START" and the format string text variable to be renamed "LABEL" and you must create a new variable in the dataset called TYPE. Since this conversion process is the bulk of the work in creating a permanent SAS format library, the remainder of the paper covers this..

4. The Case Study: Fred's Tire Shop

This section goes through a real (?) world example of how you would go about creating and storing these formats for common use.

4.1. The Data and its relationships

In our example, Fred owns a Tire Shop and is also a fledgling SAS programmer. Fred must keep track of many types of tires, the invoices surrounding them, the prices, etc. Like many types of operations, the Tire Shop has an historical collection of invoices and lookup tables.



To create the sample datasets, the following code can be used:

```
/* READ IN THE EXAMPLE DATASETS                                */
.
Data catname;
  input @1 tirecat $5. @7 catname $50.;
Cards;
A8822 VULCANIZED X-RADIAL 120-45
B8811 VINYL X-RADIAL 130-56
C4423 VULCANIZED X-RADIAL 110-30
;
run;

Data tiretype;
  input @1 tiretype 5. @7 tirecat $5. @13 tirename $45.;
Cards;
12345 A8822 ROADGRYP VULCANIZED X-RADIAL (120-45)
12367 B8811 SHURE-HOLD VINYL X-RADIAL (130-56)
11145 C4423 KORNER-KING VULCANIZED X-RADIAL (110-30)
11167 A8822 TITAN VULCANIZED X-RADIAL (120-45)
;
run;

Data detail;
  input @1 clntcd $3. @5 dtofpurc $10. @17 tiretype 5.
        @25 quantity 3. @30 cost 7. @38 sellprc 9.;
cards;
A02 03OCT1997 12345 4 101.50 199.95
A03 03OCT1997 12367 4 97.58 150.00
A02 04OCT1997 11145 4 120.50 199.95
A03 04OCT1997 11167 4 95.58 150.00
A01 04OCT1997 12345 4 101.50 199.95
A03 04OCT1997 12367 4 97.58 150.00
A01 05OCT1997 11145 4 120.50 199.95
A03 05OCT1997 11167 4 95.58 150.00
A02 05OCT1997 12345 4 101.50 199.95
A01 05OCT1997 12367 4 97.58 150.00
A01 05OCT1997 11145 4 120.50 199.95
A02 06OCT1997 11167 4 95.58 150.00
A02 06OCT1997 12345 4 101.50 199.95
A03 06OCT1997 12367 4 97.58 150.00
A02 06OCT1997 11145 4 120.50 199.95
A03 06OCT1997 99999 4 101.58 150.00
;
run;
```

To create a simple report without formats, the following code might be used (don't go through it in too much detail – this is meant to be long and boring):

```
/* EXAMPLE 1: USING LOOKUP TABLES WITHOUT FORMATS          */
.
/* STEP 1. CREATE SUMMARY FROM DETAIL */

proc sort data=detail;
  by tiretype clntcd;
run;

proc means sum noprint data=detail;
  by tiretype clntcd;
  output out=summrep sum=;
run;
```

```

/* START OF LOOKUP TABLES CODE */

/* STEP 2. ADD TIRE NAMES TO TIRETYPE */
/* NOTE: THIS IS A NICE, CLEAN EXAMPLE WHERE NO FIELD NAMES */
/*      NEED RE-NAMING OR DATA CONVERSION TO PERFORM THE MERGE */

proc sort data=summrep;
  by tiretype;
run;

proc sort data=tiretype;
  by tiretype;
run;

data summrep;
  merge summrep(in=p) tiretype;
  by tiretype;
  if p; /* KEEP TIRETYPES ONLY IF THEY HAVE PURCHASES */
run;

/* ADD TIRE CATEGORY NAMES */

proc sort data=summrep;
  by tirecat;

proc sort data=catname;
  by tirecat;

data summrep;
  merge summrep(in=p) catname;
  by tirecat;
  if p; /* KEEP TIRECATS ONLY IF THEY HAVE PURCHASES */
run;

proc print data=summrep(drop=_type_ _freq_);
  options linesize=150;
  format quantity comma6.2 cost sellprc dollar8.2;
run;

/* EXAMPLE 2. CREATING THE TIRENAME FORMAT */
/* THIS IS A NUMERIC FORMAT BECAUSE THE INPUT */
/* IS A NUMBER (TIRETYPE) */

proc format;
  value $catname /* BY USING A DOLLAR SIGN IN FRONT OF THE
                  FORMAT NAME, YOU SPECIFY THAT THE INPUT
                  WILL BE CHARACTER, NOT NUMERIC. */
    'A8822'='VULCANIZED X-RADIAL 120-45 '
    'B8811'='VINAL X-RADIAL 130-56 '
    'C4423'='VULCANIZED X-RADIAL 110-30 '
    other='UNKNOWN TIRE CATEGORY '
  ;
run;

```

```

proc format;
  value tirenm /* BY NOT USING A DOLLAR-SIGN IN THE FORMAT NAME
                YOU SPECIFY THAT THE INPUT WILL BE NUMERIC */
    12345='ROADGRYP VULCANIZED X-RADIAL (120-45)'
    12367='SHURE-HOLD VINAL X-RADIAL (130-56)'
    11145='KORNER-KING VULCANIZED X-RADIAL (110-30)'
    11167='TITAN VULCANIZED X-RADIAL (120-45)'
    other='UNKNOWN TIRE TYPE' /* OTHER IS A KEYWORD THAT WILL PRODUCE
                              A NAME FOR ALL VALUES THAT DO NOT
                              HAVE A REFERENCE */
  ;

  /* YOU CAN CREATE MORE THAN ONE FORMAT IN A PROC FORMAT */

  value tirecat
    12345='A8822'
    12367='B8811'
    11145='C4423'
    11167='A8822'
    other='?????'
  ;
run;

```

```

/* EXAMPLE 3. USING LOOKUP TABLES WITH FORMATS */

```

```

data summrep;
  set summrep;
  tirename=put(tiretype, tirenm.);
  tirecat=put(tiretype, tirecat.);
  catname=put(tirecat, catname.);
run;

proc print data=summrep(drop=_type_ _freq_);
  options linesize=150;
  format quantity comma6.2 cost sellprc dollar8.2;
run;

```

After these formats are created and stored in a permanent format library, you may use them as in the following example. Compare this to the previous Example 2.

```

/* START OF LOOKUP TABLE CODE */

data summrep;
  set summrep;
  tirename=put(tiretype, tirenm.);
  tirecat=put(tiretype, tirecat.);
  catname=put(tirecat, catname.);
run;

```

4.2 Using CNTLIN

To avoid typing, you can read in lookup tables to create your formats. As presented, the datasets generally need some transformation before they can be fed directly into CNTLIN.

```

/* EXAMPLE 4. CREATING THE FORMATS AGAIN */
/*          THIS TIME OUT OF DATASETS */

/* IN ORDER TO USE THE DATASET AS AN INPUT TO PROC FORMATS */
/* IT MUST BE SET UP TO HAVE THE FOLLOWING CHARACTERISTICS */

```

```

/* THE DATASET MUST HAVE THE FOLLOWING VARIABLES (AND OTHER) */
/* OPTIONS ARE AVAILABLE */

/*
VARIABLE    LENGTH/    DESCRIPTION
            TYPE

FMTNAME     $8.        THE NAME YOU WILL USE WHEN YOU REFER TO THE FORMAT
START       $18.       THE INPUT VALUE(S) (NOTE: EVEN NUMERICS WILL BE STORED AS
CHARACTERS)
LABEL
TYPE        $1.        THE TEXT THAT YOU WANT THE INPUT TRANSLATED TO
                        CHARACTER ('C') OR NUMERIC ('N') INPUT

SO, THE FIRST FORMAT, "$CATNAME." CAN BE SET UP AS FOLLOWS:
*/

/* EXAMPLE 4.1 USING A CONTROL DATASET FOR CHARACTER FORMATS */

/* STEP 1. GET LOOKUP TABLE INTO PROPER STRUCTURE FOR PROC FORMAT */

data fmtprep(keep= start label fmtname type);
  set catname;
  rename
    tirecat=start
    catname=label
  ;
  fmtname="$catname";
  type='C';
run;

data other; /* need to account for other values... */
  fmtname='$catname';
  start='other';
  label='UNKNOWN TIRE CATEGORY';
  TYPE='C';
run;

proc append base=fmtprep data=other;
run;

/* SINCE WE ARE GOING THROUGH THIS EFFORT - MIGHT AS WELL HOLD THESE
FORMATS PERMANENTLY. HINT: THIS PROGRAM COULD BE RUN AS A BATCH
PROCESS AND EVERYONE IN A WORKING GROUP COULD USE THE SAME FORMATS!
SAS LOOKS FOR FORMATS HELD IN A LIBREF CALLED 'LIBRARY' AUTOMATICALLY.
EVERYTIME YOU INITIATE A SAS SESSION, CREATE THE FOLLOWING LIBREF:
LIBNAME LIBRARY <PATH OF FORMAT LIBRARY>;
*/

libname library 'c:\formats';

proc format library=library cntlin=fmtprep;
run;

/* EXAMPLE 4.2 USING A LOOKUP TABLE TO CREATE A NUMERIC FORMAT */

/* THE ONLY DIFFERENCE HERE IS THAT NUMERIC FORMATS ARE TYPE 'N' */
/* AND THE START VARIABLE IS NUMERIC IN THE LOOKUP TABLE. SINCE */
/* PROC FORMATS CONVERTS THIS VALUE INTO CHARACTER, YOU CAN USE */
/* THAT TO TRICK PROC FORMAT INTO ACCEPTING 'OTHER' AS A NUMERIC */
/* VALUE - IN CREATING FMTPREP, CONVERT THE NUMERIC VALUE TO */
/* CHARACTER */

```

```

/* TIRECAT */

data fmtprep(keep= start label fmtname type);
  set tiretype;
  rename tirecat=label;
  start=left(tiretype); /* CONVERT NUMERIC TO CHARACTER */
  fmtname="tirecat";
  type='N';
run;

data other; /* need to account for other values... */
  fmtname='tirecat';
  start='other';
  label='UNKNWN';
  TYPE='N';
run;

proc append base=fmtprep data=other;
run;

proc format library=library cntlin=fmtprep;
run;

/* TIRENM */

data fmtprep(keep= start label fmtname type);
  set tiretype;
  rename tirename=label;
  start=left(tiretype); /* CONVERT NUMERIC TO CHARACTER */
  fmtname="tirenm";
  type='N';
run;

data other; /* need to account for other values... */
  fmtname='tirenm';
  start='other';
  label='UNKNWN';
  TYPE='N';
run;

proc append base=fmtprep data=other;
run;

proc format library=library cntlin=fmtprep;
run;

/* SEE IF THIS ACTUALLY WORKED... */

data summrep;
  set summrep;
  tirename=put(tiretype, tirenm.);
  tirecat=put(tiretype, tirecat.);
  catname=put(tirecat, catname.);
run;

proc print data=summrep(drop=_type_ _freq_);
  options linesize=150;
  format quantity comma6.2 cost sellprc dollar8.2;
run;

```

5. Are Formats Efficient Programming?

SAS formats are efficient in most instances. In the first place, the lookup table is held in memory. This means very fast random access, but it also means that a format is limited by the amount of memory you have. For example, if you are holding a Value Format that contains 100,000 values, you may run into memory constraints, depending on your system.

SAS formats are also efficient because of the algorithm used by the SAS engine. Formats are stored in much the same way as an index is stored so that the process is not to traverse the entire list of format values, but go directly to the format value and label (a binary search).

Finally, SAS formats are most efficient for the adhoc programmer who spends most of their time coding from scratch. As we saw in the example, coding can be done in a fraction of the time that it takes to sort, merge, or write "if... then ... else" statements. By learning how to use Proc Format and by maintaining a format library, you can make your own programming life easier.

www.pacificcoast.net/~seanburnett/sugi.doc

see the WEB site for file dropoff