PharmaSUG 2016 - Paper HT05

# Building and Using User Defined Formats

Arthur L. Carpenter
California Occidental Consultants, Anchorage, Alaska

## ABSTRACT

Formats are powerful tools within the SAS System. They can be used to change how information is brought into SAS, how it is displayed, and can even be used to reshape the data itself. The Base SAS product comes with a great many predefined formats and it is even possible for you to create your own specialized formats.

This paper will very briefly review the use of formats in general and will then cover a number of aspects dealing with user generated formats. Since formats themselves have a number of uses that are not at first apparent to the new user, we will also look at some of the broader application of formats. Topics include; building formats from data sets, using picture formats, transformations using formats, value translations, and using formats to perform table look-ups.

## KEYWORDS

format, informat PROC FORMAT, picture format, format library, FMTSEARCH, DB2

## INTRODUCTION

Formats are used to map one value into another. For instance formats are used extensively with SAS dates, which are stored as the number of days since the beginning of time (January 1, 1960). Since neither you nor your boss would like to see the date displayed as 16,014 (November 5, 2003), SAS has provided us with a number of conversion tools that allow us to store the date as a number while displaying it as a text string that we will recognize. One set of tools that change how the value is displayed are formats.

There are literally dozens of formats that SAS has created for handling dates alone. Although there are a great many formats already created it is not unusual to have a need to create a specialty format. This can be done with PROC FORMAT and there is a great deal of flexibility as to how the format is created and what it can do for you.

Formats can also be used for more than the displaying data. Formats can be combined with DATA step functions to provide a means to do data conversions and even table look-ups.

Formats are powerful and flexible. A good understanding of the use of formats is very important to a well rounded SAS programmer.

## REVIEW

There are two general classes of formats (FORMAT and INFORMAT). Informats are used when reading data into SAS and formats are used to write out values to reports and displays. Most of the discussion in this presentation applies equally to both types and a distinction will only be made when it is necessary. Although not strictly accurate most SAS programmers refer to both formats and informats as formats.

Formats are always named and the name will always include a period. The FORMAT statement is used to attach a format to one or more variables, while an INFORMAT statement is used with informats. Sample FORMAT statements could include:

```
format debits credits dollar12.2;
format ssn ssn11.;
format total dollar9. lineitem comma9.;
```

Because we have both numeric and character variables we also need numeric and character formats. Character format names are preceded with a $. A few selected formats include:

- dollar*w.d* includes dollar sign and commas
- percent*w.d* writes the number as a percent
- ssn*w.* converts a number to a social security number
- *w.d* where *w* is the width and *d* the number of decimal places
- z*w.d* writes leading zeros
- $char*w.* writes standard character data preserving leading blanks
- $*w.* writes standard character data

The application of these formats to the data values on the left could produce the following results:

```
2345.678    --->  dollar9.2    ---> $2,345.68
0.6723      --->  percent8.2   --->  67.23%
-0.6723     --->  percent8.2   ---> (67.23%)
123456789   --->  ssn11.       ---> 123-45-6789
2345.678    --->  10.2         --->    2345.68
2345.678    --->  z10.3        ---> 002345.678
'  abcde'   --->  $char8.      --->   '  abcde'
'  abcde'   --->  $8.          --->   'abcde  '
```

## CREATING OUR OWN FORMATS

Although SAS provides a large number of ready made FORMATS and INFORMATS, it is often necessary to create formats designed for a specific purpose. Formats are created using PROC FORMAT. User defined formats can be used to:

- convert numeric variables into character values
- convert character strings into numbers
- convert character strings into other character strings
- display values in special ways

PROC FORMAT features include:

- format definition through the VALUE and INVALUE statements
- creation of template style (picture) formats
- formats created from the contents of a data set
- data sets created from formats
- permanent storage and sharing of formats

The FORMAT procedure is fairly straightforward for simple formats, however there are a number of seldom used options that provide a great deal of power and flexibility. The general syntax of the procedure is:

```
PROC FORMAT options;
VALUE format_name specifications;
INVALUE informat_name specifications;
PICTURE format_name specifications;
RUN;
```

The format name is a valid SAS name, with character formats starting with $. In the VALUE, INVALUE, and PICTURE statements, the specifications are made in value pairs. These pairings are in the form of:

```
incoming_value = formatted_value
```

## USING THE VALUE STATEMENT

Simple formats are created using the VALUE statement. It includes the name of the format to be created and the paired mapping of values (on the left of the = sign) and what those values will be mapped to (on the right of the = sign). The following example creates a format ($region.) that maps values of a character variable that ranges from '1' to '9'. Since the LIBRARY= option is specified, the format will be stored permanently in a catalog named LIBRARY.FORMATS, which in this case will be stored in the c:\junk location (a particularly good name for a permanent location - don't you think?).

```
libname library 'c:\junk';

proc format library=library;
value $region
   '1'   = 'group 1'
 '2','5' = 'group 2'
 '3','4' = 'group 3'
 '6'-'9' = 'Western'
  other  = 'miscoded'
;
run;

proc print data=sasclass.clinics(obs=7);
```

```
var region lname fname;
format region $region.;
title1 'Clinics data using the $region format';
run;
```

Clinics data using the $region format

| OBS | REGION | LNAME | FNAME |
|---|---|---|---|
| 1 | group 3 | Smith | Mike |
| 2 | group 3 | Jones | Sarah |
| 3 | group 2 | Maxwell | Linda |
| 4 | Western | Marshall | Robert |
| 5 | miscoded | James | Debra |
| 6 | group 1 | Lawless | Henry |
| 7 | Western | Chu | David |

### Format Types

Formats can be applied to both numeric and character variables, however a given format can only be used for one or the other. The type of variable that the format is to be used with is determined when the format is created. Character format names start with a dollar sign ($) and the incoming values to be mapped are quoted.

### Format Specification Assignment Mappings

Assignments are made by forming pairings in the VALUE (as well as INVALUE and PICTURE) statement. The pairs are linked with an equal sign (=) and the incoming value specifications can have a number of forms:

```
single value           1       = 'Jan'
list of values      '1' , '2'  = 'acceptable values'
value range         1 - 12     = 'pre-teen'
exclusive range     1 - <100   = 'under 100'
exclusive range 12.99 <- 19.99 = 'teenager'
out of range        other      = 'miscoded'
extreme value       low - 0    = ' non-positive'
extreme value       100 - high = 'centenarian'
```

## TABLE LOOK-UPS

When you use the value of one variable to determine the value of another, you have performed a table look-up. Just as you would use a friends name to look-up their phone number in the telephone book, you can use a format to look-up a value by using the value stored in another variable.

Beginning programmers will often use IF-THEN or IF-THEN-ELSE statements to perform this type of conversion. If you have a format that maps one value to another you can use that format to modify data values. Value conversions are made by combining formats with the PUT function thus avoiding IF-THEN-ELSE processing. The result is easier to code and faster to run. The following example converts the value of the variable REGION into groups. In this example the conversion is done two ways; with IF-THEN-ELSE statements and with a PUT function.

```
data clin; set sasclass.clinics;
length group fmtgrp $8;
keep lname fname group fmtgrp region;

* create GROUP using IF-THEN-ELSE processing;
if region='1' then group = 'group 1';
else if region in('2','5') then group = 'group 2';
else if region in('3','4') then group = 'group 3';
else if '6' le region le '9' then group = 'Western';
else group = 'miscoded';

* create FMTGRP using a table lookup;
fmtgrp = put(region,$region.);
run;

proc print data=clin(obs=7);
```

```
var region group fmtgrp lname fname;
title1 'Using table lookup';
run;
```

Clearly the use of the PUT function to create the variable FMTGRP simplifies the look-up process.

Using table lookup

| OBS | REGION | GROUP | FMTGRP | LNAME | FNAME |
|---|---|---|---|---|---|
| 1 | 3 | group 3 | group 3 | Smith | Mike |
| 2 | 3 | group 3 | group 3 | Jones | Sarah |
| 3 | 2 | group 2 | group 2 | Maxwell | Linda |
| 4 | 7 | Western | Western | Marshall | Robert |
| 5 | 10 | miscoded | miscoded | James | Debra |
| 6 | 1 | group 1 | group 1 | Lawless | Henry |
| 7 | 9 | Western | Western | Chu | David |

## BUILDING A PICTURE FORMAT

Picture formats use a template of predefined characters to place the value. The template is built using the PICTURE statement. The template is built using combinations of the numbers 0, 9, and text characters. Zeros (0) are used as place holders and nines (9) are used to designate positions that must be filled with values. The following example creates a temporary format named PHONE. that will take a numeric telephone number, with or without an area code.

```
proc format;
picture phone
  2000000-9999999     = '999-9999'
  9999999<-9999999999 = ' 999) 999-9999' (prefix='(')
  other               = 'miscoded';
run;

data numbers;
type='voice'; number=    7589245; output;
type='fax  '; number= 9197834017; output;
type='modem'; number=16037619434; output;
run;

proc print data=numbers;
title1 'Telephone numbers';
format number phone.;
run;
```

Telephone numbers

| OBS | TYPE | NUMBER |
|---|---|---|
| 1 | voice | 758-9245 |
| 2 | fax | (919) 783-4017 |
| 3 | modem | miscoded |

An alternate PROC FORMAT might be written as:

```
proc format;
picture phone
  2000000-9999999999 = '000 999-9999'
  other              = 'miscoded';
run;
```

The following picture statement creates the format TONS. which can be used to display non-negative values (>= 0) that vary widely in magnitude. The problem is that when the number is small (0 to 10) we want to show two decimal places, however as the number increases, we need to show fewer decimal places, and then for values over 999 commas should be included.

```
proc format;
```

```
picture Tons
          0              =  '9'❶
         0< -      <1    =  '99' (prefix='0.' mult=100)❷
          1 -      <10   =  '9.00' ❸
         10 -     <100   =  '00.0' ❹
        100 -    <1000   =  '000.0'
       1000 -   <10000   =  '0,000' ❺
      10000 -  <100000   =  '00,000'
     100000 - <1000000   =  '000,000'
    1000000 - high       =  '000,000,000';
run;
```

❶ The value of 0 is displayed as 0. The '9' acts as a single digit placeholder to indicate that a number is required.

❷ In earlier versions of SAS values between 0 and 1 were problematic as the PICTURE statement had trouble handling values in this range. The PREFIX= option specifies the leading characters to use, including the decimal point. Since the decimal point is supplied the fractions are converted to integers by multiplying them by 100.
In the current version of SAS no special handling is required. Here all values between 0 and 10 are displayed with two decimals.

```
0< -      <10    =  '9.99' ❷
```

❸ The numbers from 1.00 to just less than 10 are displayed with two decimal places and a single leading digit. The '9' is a place holder and the 0 indicate that a number should be placed in that position if available.

❹ Write out values between 10 and 100 with one decimal. This pairing could have been written as:

```
10 -     <100   =  '99.0'
```

❺ Values larger than 1000 are displayed without decimals and include commas as appropriate. This and the following pairings could be combined as:

```
1000 - high       =  '000,000,000';
```

The PICTURE statement also supports the DATATYP= option. This option is used to specify a date, and can take on the values of: date, datetime, time.

This option allows the use of 'directives', which tell the PICTURE statement how to further structure or format the value. The following format is used to rewrite a SAS datetime value to the form used in DB2 tables.

```
proc format;
picture dbdate
   other = '%Y-%0m-%0d:%0H:%0M:%0S' (datatype=datetime);
   run;

data _null_;
  now = '05nov2003:10:30:27'dt;
  put now=;
  put now= dbdate.;
run;
```

The LOG shows:

```
now=1383647427
now=2003-11-05:10:30:27
```

There are over 15 directives and the case of the letters used to signify the directives is important:

| | |
|---|---|
| Y | year |
| m | month |
| d | day |

| | |
|---|---|
| H | hour |
| M | minute |
| S | second |
| b | month abbreviation |
| B | month name |

For the same datetime value as used above, the following formats can be used to display all or part of the month name.

```
proc format;
picture monabb
   other = '%b '       (datatype=datetime);
picture moname
   other = '%B       ' (datatype=datetime);
```

```
171  data _null_;
172    now = '05nov2003:10:30:27'dt;
173    put now=;
174    put now= dbdate.;
175    put now= monabb.;
176    put now= moname.;
177  run;

now=1383647427
now=2003-11-05:10:30:27
now=NOV
now=November
```

The DATATYPE= option could also be DATE or TIME. Be sure to leave sufficient space in the quoted string for the resolved value.

## BUILDING FORMATS FROM DATA

Formats can be built directly from any SAS data set that has two variables that contain the two parts of the format definition pairings. Once this format has been created, the storage requirements for data sets that contain long variables with often repeated values can be substantially reduced by only storing a code and a format. The data set SASCLASS.CLINICS contains both a clinic number and a clinic name, however storing the name becomes unnecessary if we create a format that will supply the name given the clinic number.

First a specialized data set is created that will supply the pairings to PROC FORMAT which expects specific variables to be present. Although additional variables can be used, the following example has the minimum that MUST be present.

```
data control(keep=fmtname start label);
set sasclass.clinics(keep=clinnum clinname);
retain fmtname '$clname'; ❶
rename clinnum=start ❷ clinname=label ❸;
run;

proc sort data=control nodupkey;
by start;run;

proc format cntlin=control; ❹
run ;

proc print data=sasclass.clinics (obs=7);
var clinnum lname fname;
format clinnum $clname.;
title1 'Clinic number formatted with a data defined format';
run;
```

```
   Clinic number formatted with a data defined format

OBS             CLINNUM              LNAME       FNAME

  1    Bethesda Pioneer Hospital     Smith       Mike
  2    Naval Memorial Hospital       Jones       Sarah
  3    New York General Hospital     Maxwell     Linda
  4    Kansas Metropolitan           Marshall    Robert
  5    Seattle Medical Complex       James       Debra
  6    Vermont Treatment Center      Lawless     Henry
  7    San Francisco Bay General     Chu         David
```

```
        Control Data Set Used to Create $CLNAME.
             ❷              ❸                                  ❶
OBS    START     LABEL                          FMTNAME

  1    011234    Boston National Medical        $clname
  2    014321    Vermont Treatment Center       $clname
  3    023910    New York Metro Medical Ctr     $clname
  4    024477    New York General Hospital      $clname
  5    026789    Geneva Memorial Hospital       $clname
  6    031234    Bethesda Pioneer Hospital       $clname
```

❶ The format name is stored in the variable FMTNAME.

❷ The left side of the assignment pairing is stored in the variable START.

❸ LABEL is used to store the right side of the assignment pairing.

❹ The CNTLIN= option is used to identify the data set that has the format name(s) and pairings.

## BUILDING DATA FROM FORMATS

When formats already exist, they can be used to create data sets by using the CNTLOUT= option in PROC FORMAT. This is essentially the opposite of using data to create the format, and the resulting data set will contain virtually all the variables that can be used to show what other variables can be used by PROC FORMAT when building formats from data sets.

```
proc format library=work
            cntlout=tons(where=(fmtname='TONS'));
run;
```

This is actually an excellent way to 'learn' what variables are used in a control data set. Find a complicated format, create the control data set and study it. Mimic what it does to build your own complicated control data set for your own frormat.

## FORMAT SEARCHES

Formats are saved in a catalog (usually with the name FORMATS). When requesting a user defined format, SAS first checks in WORK.FORMATS and then, if the *libref* LIBRARY is defined, SAS will look in LIBRARY.FORMATS. Since formats are not usually conveniently located in these two locations, we need to be able to search for formats in a variety of places and in catalogs named something other than FORMATS.

### Using FMTSEARCH=

The FMTSEARCH= system option is used to identify the *libref*s and order for the search. In the following example SAS will look for the format in the catalog PROJ1.MYFMTS and then in catalogs named FORMATS in the *libref*s WORK and LIBRARY in that order. Since WORK appears in the FMTSEARCH list, the default catalog is no longer WORK.FORMATS, and since WORK is not listed first, it is not searched first.

```
libname library 'c:\sasclass\data';
libname proj1   'd:\client3\project1\format';

options fmtsearch=(proj1.myfmts work library);
```

### Using Concatenated Catalogs

Catalogs with the same name will be implicitly concatenated when they reside within concatenated libraries. In the following example formats are being written to two different libraries (the *libref*s are OLDFMT and NEWFMT).

```
libname oldfmt v8 'c:\junk1';
libname newfmt v8 'c:\junk2';
libname allfmt v8 (newfmt oldfmt);

proc format library=oldfmt;
  value yesno 1 = 'Yes'
              0 = 'No';
  run;

proc format library=newfmt;
  value gender 1 = 'Female'
               0 = 'Male';
  run;

title1 Display Format names;
proc catalog cat=allfmt.formats;
  contents;
  run;
```

PROC CATALOG is used here to show the locations of the formats. In the following output, the column LEVEL refers to the library containing the catalog which contains the format.

Display Format names

Contents of Catalog ALLFMT.FORMATS

| # | Name | Type | Level | Create Date | Modified Date |
|---|---|---|---|---|---|
| 1 | GENDER | FORMAT | 1 | 20JUL2003:09:54:41 | 20JUL2003:09:54:41 |
| 2 | YESNO | FORMAT | 2 | 20JUL2003:09:54:40 | 20JUL2003:09:54:40 |

## USING MULTI-LABEL FORMATS

Overlapping range values are now allowed for formats, but only selected procedures are able to utilize these MULTI-LABEL formats. The formats are created using the MULTILABEL option.

```
proc format;
value wtgrp (multilabel)
   0-<120    = ' <120'
   120-200   = '120-200'
   200<-high = 'over 200'
   220<-high = 'over 220';
run;


proc tabulate data=sasclass.clinics;
  class wt / mlf;
  table wt all,n='Count';
  format wt wtgrp.;
  run;
```

| | Count |
|---|---|
| weight in pounds | |
| <120 | 16 |
| 120-200 | 52 |
| over 200 | 12 |
| over 220 | 2 |
| All | 80 |

Procedures that do not utilize overlapping format values will only use the primary range. In this case the primary is 'over 200' and the secondary is 'over 220'. This is because 'over 200' is first alphabetically and **NOT** because it was defined first.

## USING NESTED FORMAT CALLS

Although nested format calls are not strictly related to user defined formats, I just have to at least mention them here. You can create a format that calls another format. Perhaps you have a format that almost does what you want but needs a minor modification. The DATE9. format for instance displays a SAS date in the DDMONYYYY form, however missing values are displayed as a dot (.). I want my DATE9. format to display the word UNKNOWN instead of a dot.

```
proc format ;
   value mydate
      . = 'Unknown'
      other = [date9.];
   run;
title1 'Showing unknown Dates';
proc print data=sasclass.clinics;
   var fname lname dt_diag admit disch;
   format admit dt_diag disch mydate.;
   run;
```

| Obs | fname | lname | dt_diag | admit | disch |
|---|---|---|---|---|---|
| 1 | Mike | Smith | 14FEB1987 | 14FEB1987 | 15FEB1987 |
| 2 | Sarah | Jones | 03JUL1983 | 05JUL1983 | 10JUL1983 |
| 3 | Linda | Maxwell | 03JUL1983 | 05JUL1983 | 10JUL1983 |
| 4 | Robert | Marshall | Unknown | Unknown | Unknown |
| 5 | Debra | James | 03MAY1983 | 27JUL1985 | 03AUG1985 |
| 6 | Henry | Lawless | 05NOV1986 | 05NOV1986 | 19NOV1986 |

In the format MYDATE., the missing value maps to 'Unknown' and the 'other' keyword maps all other values to a nested format (DATE9.). Notice that the nested format is in square braces and is not quoted.

In SAS9.4 you can even call single argument functions from within a format. The format MYSQRT. will display the square root of the incoming value.

```
proc format ;
   value mysqrt
      other = [sqrt()];
   run;
```

Using this format to display a value will cause the display of the square root of the value instead of the actual stored value. This can save a transformation step.

```
proc print data=sashelp.class(obs=5) ;
   var name age height;
   format height mysqrt.;
   run;
```

| Obs | Name | Age | Height |
|---|---|---|---|
| 1 | Alfred | 14 | 8.3066 |
| 2 | Alice | 13 | 7.5166 |
| 3 | Barbara | 13 | 8.0808 |
| 4 | Carol | 14 | 7.9246 |
| 5 | Henry | 14 | 7.9687 |

This technique can be especially helpful when you want to apply a format to a continuous variable. Even better you can create your own specialized function using PROC FCMP.

## PRELOADING USER DEFINED FORMATS

The REPORT, MEANS, SUMMARY, and TABULATE procedures have the ability to use the PRELOADFMT option to preload formats prior to the execution of the procedure. Through the use of this and related options, you can filter observations (like a WHERE clause), as well as, add rows to your table that are not represented in your data (sparsing), While the PRELOADFMT option will always be present, the other supplemental options vary by procedure.

The format $SYMP shown here specifies a label for two of the 10 symptom codes. It also specifies a 'Bad Code' (00), which never appears in the data.

```
proc format;
   value $symp
      '01' = 'Sleepiness'
      '02' = 'Coughing'
      '00' = 'Bad Code' ;
   value $gender
      'M','m' = 'Male'
      'F','f' = 'Female'
      ' '     = 'Unknown';
   run;
```

Applying $SYMP. in a PROC TABULATE step, along with the PRELOADFMT and EXCLUSIVE options on the CLASS statement for SYMP, demonstrates how powerful the use of preloaded formats can be.

```
proc tabulate data=sasclass.clinics;
   class sex  / preloadfmt
                exclusive
                missing;
   class symp / preloadfmt
                exclusive;
   var wt;
   table (sex all)*wt=' '*n=' '
         ,symp all
         /box='Patient Counts'
          row=float
          misstext='0' printmiss;
   format sex $gender. symp $symp.;
   run;
```

| Patient Counts | symptom code | | | All |
|---|---|---|---|---|
| | Bad Code | Sleepiness | Coughing | |
| patient sex | | | | |
| Female | 0 | 2 | 6 | 8 |
| Male | 0 | 2 | 4 | 6 |
| All | 0 | 4 | 10 | 14 |

Inspection of the resulting table shows that symptoms not in the format are eliminated (like with a WHERE clause), and a column for the 'BAD CODE' has been added even though it is not in the data (you cannot do that with a WHERE clause!).

## SUMMARY

Formats are extremely useful to the SAS programmer as they provide both power and flexibility. A large number of formats have been created by SAS and are provided to all users with the Base language.

The Base language also includes PROC FORMAT which can be used to create customized formats so that you will have the correct format when the provided formats are not 'just right'. This procedure is fairly easy to use and the formats that are created can be stored permanently where they can be made available to all the users in the group.

Format types include the newer mult-label formats and the underutilized picture formats. Picture formats have the ability to create formats based on a template that includes a number of options and 'directives' tailored for use with SAS date and datetime values.

## ABOUT THE AUTHOR

Art Carpenter's publications list includes five books, and numerous papers and posters presented at SAS Global Forum and other SAS user conferences. Art has been using SAS® since 1977 and through California Occidental Consultants he teaches SAS courses and provides contract SAS programming support nationwide.

## AUTHOR CONTACT

Arthur L. Carpenter
California Occidental Consultants
10606 Ketch Circle
Anchorage, AK 99515

(907) 865-9167
art@caloxy.com
www.caloxy.com





## REFERENCES

Morgan, Derek P., 2014, *The Essential Guide to SAS® Dates and Times, Second Edition*, SAS Press, SAS Institute Inc., Book number #66300.
http://www.sas.com/store/books/categories/examples/the-essential-guide-to-sas-dates-and-times-second-edition/prodBK_66300_en.html

More on preloaded formats can be found at:

Carpenter, Arthur L., 2008, "The MEANS/SUMMARY Procedure: Getting Started and Doing More", presented at the 16th Annual Western Users of SAS Software, Inc. Users Group Conference (WUSS), Universal City, CA. Also presented at the PharmaSUG conference, 2009 (paper TT05 and TT06).
http://support.sas.com/resources/papers/proceedings10/135-2010.pdf

Carpenter, Arthur L., 2010, "PROC TABULATE: Getting Started and Doing More". Presented in 2010 at the Western Users of SAS Software Conference , WUSS, and also in 2010 at the Michigan SAS User Group Conference, South Eastern SAS Users Group, SESUG, Midwestern SAS User Group, MWSUG, and at the Pharmaceutical SAS User Group, PharmaSUG. Presented in 2011 at the SAS Global Users Group Conference. The paper appears in the proceedings of each of these conferences.
http://support.sas.com/resources/papers/proceedings11/173-2011.pdf

Carpenter, Arthur L, 2012, "Programming With CLASS: Keeping Your Options Open". Published in the conference proceedings for PharmaSUG 2012, WUSS 2012, MWSUG 2012, WUSS 2014.
http://www.pharmasug.org/proceedings/2012/TA/PharmaSUG-2012-TA10.pdf

## TRADEMARK INFORMATION