Proc Format Uncovered

Amadeus Software Limited

ABSTRACT

Most SAS programmers encounter the need for Proc Format very early on in their SAS programming careers: It doesn't take long to realise that although SAS provides a Dollar format, unfortunately there is no equivalent Pound format. This discovery invariably sparks the creation of some user-defined format using the format procedure.

Unfortunately for many programmers, the understanding of the format procedure ends with the value statement, without them ever realising the full potential of this powerful procedure. Other capabilities of the procedure include printing the contents of a formats catalog, storing format and informat definitions in a dataset and creating formats and informats from a dataset.

This paper uncovers some of these lesser-known capabilities of PROC FORMAT, exposing picture formats, nested formats, multilabel formats, CNTLIN / CNTLOUT datasets, the formats catalog and several useful options along the way.

INTRODUCTION

The format procedure is very different to other base SAS procedures: It's main purpose is not to generate output, calculate any statistics, summarise your data, directly alter your data sets in any way, nor is it similar to any of the utility procedures. It can be used in isolation, completely separately from your data sets, most commonly to define your own formats and informats ready for use in any subsequent data or proc step.

A basic knowledge of the value and invalue statements from PROC FORMAT, coupled with the concept of SAS formats and informats will be assumed. This paper aims to take you beyond these basics, equipping you with the know-how to master this powerful subsystem within the Base SAS product.

In order to ease the reading of this paper I will be using the term "format" to refer to both formats and informats wherever possible, distinguishing between the two where applicable. All coded examples used in this paper have been run using SAS V8.2.

DID YOU KNOW?…

Before covering some more specific uses of the format procedure, there are some general concepts for us to be aware of first:

FORMAT NAMES

The standard SAS naming conventions apply when specifying names for our formats, with the following exceptions and additional restrictions:

1. The maximum permissible length of a format name is just 8 characters when using SAS V8.2 or earlier, whereas up to 32 characters are permissible in SAS V9.
2. All names for character formats and character informats must start with the dollar symbol ($). Note that this dollar symbol does take up one of the permissible characters. (A character format expects character *input*, whereas a character informat produces character *output*)
3. There is an implied @ symbol on the front of all informat names. Note that this @ symbol also takes up one of the permissible characters.
4. Format names cannot end in a number. This would confuse the issue of specifying a format width when applying a format to a variable.

Permissible length of format names		Allow For	SAS V8.2 or earlier	SAS V9
Format Names	Numeric		8	32
	Character	$	7	31
Informat Names	Numeric	@	7	31
	Character	@ and $	6	30

It should be noted that no errors are generated if we try to create a format with a name that is too long:

```
proc format;
  value $NameIsTooLong
    '1' = 'M'
    '2' = 'F';
run;
```

The format is still created with the format name truncated. A note appears in the log:

```
56   proc format;
57         value $NameIsTooLong
NOTE: The format name '$NAMEISTOOLONG' exceeds 8 characters.
      Only the first 8 characters will be used.
58           '1' = 'M'
59           '2' = 'F';
NOTE: Format $NAMEIST has been output.
60       run;
```

If we try to use this format in a subsequent data or proc step, again the format name is truncated, however this time a warning message appears in the log:

```
61      data example1;
NOTE: SCL source line.
62          format gender $NameIsTooLong.;
                          ---------------
                          503
WARNING 503-185: The format name $NAMEISTOOLONG exceeds 8
                 characters, and will be truncated.
```

This truncation of long format names can be dangerous when using SAS V8.2 or earlier. For instance, formats will be overwritten if we create several formats with long names whose first 8 characters are not unique. Be aware that only a note in the log will point this out to you.

One final thing to know is not to add a period to the end of the format name when creating the format in PROC FORMAT. The period is only added to the end of the format name when we want to *apply* the format, for example when using the PUT function:

```
gender=put('1',$NameIsTooLong.);
```

It is the period that distinguishes a format name from a variable name.

RANGES AND VALUES

After coming up with a name for a user-defined format, ranges and formatted values must be specified. Ranges are the inputs to the format; the values you want to convert. Formatted values are the outputs of the format; what you want the ranges to become.

```
proc format;
  value $example
    'Range 1' = 'Formatted Value 1'
    'Range 2' = 'Formatted Value 2'
    'Range 3' = 'Formatted Value 3';
run;
```

The formatted values are always character strings, regardless of whether you are creating a numeric or character format, and can be up to 32,767 characters. Strictly speaking the ranges should not overlap since this will normally result in error messages. However, you are able to get away with 'touching' range values where the end value of one range equals the start value of another range:

```
proc format;
  value OverlapA
    10-20='Range 1'
    20-30='Range 2';
run;
```

In the OVERLAPA format we can see that the number 20 is the end value for range 1 whilst also being the start value for range 2. This format will not generate any error messages. When range values meet in this fashion PROC FORMAT uses the formatted value from the first range. Therefore the number 20 will be formatted as 'Range 1' using the OVERLAPA format. To eliminate this ambiguity involved with touching range values it is good practice to exclude values from a range by using the "<" symbol:

```
proc format;
value OverlapB
    10 -<20 ='Range 1'
    20<- 30 ='Range 2'
    30<-<40 ='Range 3';
run;
```

In the OVERLAPB format the "-<" notation in range 1 has been used to exclude the end value of 20 from the range. Similarly the "<-" notation in range 2 will exclude the start value of 20 from the range. Finally the "<-<" notation in range 3 has been used to exclude both the starting value of 30 and the end value of 40 from the range.

The LOW and HIGH keywords can be used in your range specifications to capture the lowest possible value and the highest possible value respectively. Note that missing values, although taking on a value smaller than any other number or printable character, are not included in the range when the LOW keyword is used. The OTHER keyword can be used on its own in a range specification to capture any other values, including missing values, which have been excluded from all the other ranges.

```
proc format;
  value KeywordA
    low -< 10   ='Lower than 10, not missing'
    10  -  high ='10 and higher'
    other       ='Missing value';

  value KeywordB
    low-high='everything except missing values';

  value KeywordC
    other   ='everything including missing values';
run;
```

When using the INVALUE statement to define your own informat, you will similarly need to define ranges and *informatted* values (the value you want the raw data to become). There are two useful keywords available for use as informatted values:

- _ERROR_
  Will treat data values in the defined range as invalid data. SAS will assign a missing value to the variable, print the raw data line to the SAS log and display a warning message

- _SAME_
  Will stop the informat converting the raw data to any other value i.e. leaves the raw data untouched

```
proc format;
  invalue Keyword
    9999    =   _ERROR_
    1-100   =   _SAME_
    other   =   . ;
run;
```

SOME USEFUL OPTIONS TO KNOW

This section outlines some useful options that can be used on the INVALUE, VALUE and PICTURE statements. These options need to be declared in parentheses after the format name and will affect the entire format being created.

THE FUZZ FACTOR

The FUZZ= option can be used on the VALUE or INVALUE statements to define a fuzz factor for matching values to a range. If a number does not fall within a range exactly, but falls within the fuzz factor of that range, then the format considers this as a match.

```
proc format;
  value fuzzyA (fuzz=0.1)
    5 = 'Average'
    6 = 'Good'
    7 = 'Excellent';
run;
```

Here the fuzz factor of 0.1 means that any number falling within 0.1 of the range values will have the corresponding formatted value applied to it. For example, the number 6.95 will be formatted as 'Excellent'. In fact the following format without the FUZZ= option would yield equivalent formatted results:

```
proc format;
  value fuzzyB
    4.9 - 5.1 = 'Average'
    5.9 - 6.1 = 'Good'
    6.9 - 7.1 = 'Excellent';
run;
```

The default fuzz factor is 1E-12 for numeric formats and 0 for character formats. Therefore we can save storage space by specifying FUZZ=0 when using the VALUE statement to create numeric formats.

If the value of a variable matches one range without the help of a fuzz factor and another range with the help of the fuzz factor, then it will be formatted using the value that it matched without the fuzz factor.

The fuzz factor has no effect on any values excluded from a range using the < operator. Such an excluded value will not be formatted with the corresponding value even if it falls into the range with the help of a fuzz factor.

```
proc format;
  value fuzzyc (fuzz=0.25)
    1 - 2       = 'Acceptable'
    3 <- 4      = 'Not Acceptable'
    other       = 'Unknown';
run;
```

Using the FUZZYC format created above, the number 3 will be formatted as 'Unknown' since it is excluded from the range by the < operator and is therefore ignored by the fuzz factor. However, the number 2.9, being within 0.25 of 3, will be formatted as 'Not Acceptable' due to the fuzz factor.

NOTSORTED OPTION

By default, the ranges of a format are stored in sorted order. Alternatively, we may want to enforce that the ranges are stored in the order that they are defined. This can be done by using the NOTSORTED option. This can prove useful in the following circumstances:

1. If we know that certain ranges occur more frequently than others then we could declare these ranges first when creating our format and use the NOTSORTED option. Then when applying our format, the more frequent ranges will be searched first, saving processing time
2. We may wish to preserve the order in which the format ranges were defined when printing our format using the FMTLIB option (see later for this option)
3. We may want to retain the order of range definition when using the ORDER=DATA and PRELOADFMT options when analysing class variables using Proc Means, Proc Summary or Proc Tabulate

JUST OPTION

This option can only be used on the INVALUE statement when creating an informat. The JUST option will left-justify the input values before comparing them to the informat ranges. This can be very useful when the length of the variable is longer than the length of the informat ranges.

```
proc format;
  invalue $jstat (just)
    'M' = 'Married'
    'S' = 'Single'
    'D' = 'Divorced';
run;
```

If the $JSTAT informat were applied to a variable of length 3, then the JUST option would enable the character strings "M ", " M " and " M" all to be formatted as 'Married'. Without the use of the JUST option, only "M " would be formatted as 'Married'.

UPCASE OPTION

This option can only be used on the INVALUE statement when creating an informat. The UPCASE option will convert the input values to uppercase before comparing them to the informat ranges.

```
proc format;
  invalue $ustat (upcase)
    'M' = 'Married'
    'S' = 'Single'
    'D' = 'Divorced';
run;
```

If the $USTAT informat were applied to the character strings "d" and "D", then both character strings would be formatted as 'Divorced'.

FORMAT CATALOGS

The user-defined formats created by PROC FORMAT are stored in SAS catalogs, by default the FORMATS catalog in the WORK library (WORK.FORMATS).

STORING FORMATS PERMANENTLY

As we all know, the WORK library is only a temporary storage location, therefore any formats stored in this FORMATS catalog are lost when we end our SAS session. To be able to use our formats in subsequent SAS sessions we need to store our formats in a permanent location, which can be done using the LIBRARY= option on the PROC FORMAT statement.

```
proc format library=libref<.catalog>;
```

From the above syntax, we can see that specifying a catalog name is optional. If the catalog name is omitted then the formats will be created in `libref.FORMATS`.

TELL SAS WHERE TO FIND YOUR FORMATS

Merely storing our formats in a permanently library isn't sufficient for SAS to be able to find them. We actually have to tell SAS where to look to find our formats by using the FMTSEARCH system option. This option controls which catalogs SAS should search to find our formats, and in what order. By default the WORK.FORMATS catalog is always searched first, followed by the LIBRARY.FORMATS catalog, unless one of them appears in the FMTSEARCH= list. As before, if a catalog name is omitted, then the FORMATS catalog name is used. This option can either be set in your configuration file, at SAS invocation time, using a global options statement, or by using the system options window.

```
/* 1 */ options FMTSEARCH=(views mylib.mycat);
/* 2 */ options FMTSEARCH=(views work mylib.mycat library);
```

In the above example, SAS will search the format catalogs in the following order:

Order	Example 1	Example 2
1	WORK.FORMATS	VIEWS.FORMATS
2	LIBRARY.FORMATS	WORK.FORMATS
3	VIEWS.FORMATS	MYLIB.MYCAT
4	MYLIB.MYCAT	LIBRARY.FORMATS

NOFMTERR OPTION

If we assign a format to a variable during a data step, then that format is permanently attached to that variable and will be used automatically thereafter. The disadvantage of this is that in order to now use this data set, the format must always be available to us. If we attempt to use a data set without having the required format accessible, then SAS generates an error message, preventing us from using the data in any way.

```
ERROR: Format $G not found or couldn't be loaded for variable gender.
NOTE: The SAS System stopped processing this step because of errors.
```

To inhibit this behaviour, and to enable us to continue using a data set with inaccessible formats, we must use the NOFMTERR system option. This will substitute the default w. or $w. formats in place of any inaccessible formats, issue a note to the log, and continue processing as normal.

```
NOTE: Format $G was not found or could not be loaded.
NOTE: There were 104 observations read from the data set WORK.EXAMPLE2.
```

PRINTING FORMAT CATALOG ENTRIES

The FMTLIB option can be used on the PROC FORMAT statement to request that SAS print information about all the formats and informats stored in the catalog specified in the LIBRARY= option, to the output window. If you don't want to print all the formats from the specified catalog, then you can selectively choose the ones you do want by using the SELECT and EXCLUDE statements. Note that you cannot use both the SELECT and EXCLUDE statements in the same PROC FORMAT step. In fact you can only include one SELECT statement or one EXCLUDE statement per step. When specifying the names of your formats and informats on these statements you will need to use the appropriate prefix as

discussed previously: $ symbol prefix for character formats and informats, @ symbol prefix for all informats.

```
proc format library=work.formats fmtlib;
  select  n_fmt      /* numeric format */
          $c_fmt     /* character format */
          @n_infmt   /* numeric informat */
          @$cinfmt   /* character informat */;
run;
```

PICTURE FORMATS

Picture formats can be thought of as templates for changing the way numerical data is displayed. The formatted values of formats created with a VALUE statement are fixed, and often merely thought of as labels. In contrast, the formatted values of formats created with a PICTURE statement will vary depending on the number being input.

```
proc format;
  picture years
    low - high = '000 yrs';
run;
```

For example, when the YEARS picture format is applied to a numeric variable containing ages, the formatted value will vary as the values of the age variable vary. For instance, 15 will be formatted as 15 yrs, 65 will be formatted as 65 yrs, and so on.

A picture template is a sequence of characters (maximum 40) enclosed in single quotation marks. There are three different types of characters that can be used in picture templates: digit selectors, message characters and directives.

DIGIT SELECTORS

Digit selectors are numeric characters (0 to 9 inclusive), which are placeholders for numbers. A maximum of 16 digit selectors can be used in a picture template.

There are two different types of digit selectors:

1. 0 (zero)
   The zero is a conditional placeholder. This means that if there is a digit in this position then it will be printed, otherwise this position will be empty in the formatted value.

2. Any non-zero number
   Any non-zero number is an absolute placeholder (however the number 9 is most often used). This means that if there is a digit in this position then it will be printed, otherwise this position will be filled with a 0 (zero) in the formatted value.

```
proc format;
  picture PictA low - high = '00000';
  picture PictB low - high = '99999';
  picture PictC low - high = '12345';
  picture PictD low - high = '00999';
  picture PictE low - high = '009.00';
run;
```

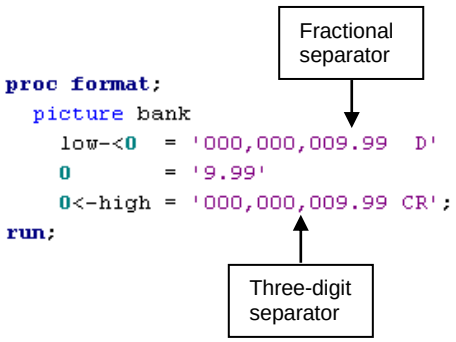
For example, the number 42 formatted using the…

- PICTA format would be formatted as 42
- PICTB format would be formatted as 00042
- PICTC format would be formatted as 00042
  (don't forget that any non-zero number is an absolute placeholder)
- PICTD format would be formatted as 042
- PICTE format would be 42.00

Notice that there are two decimal places printed here, even though we used conditional digit selectors in these positions. This happens because any digit selectors placed after the decimal point in a picture template are always treated as absolute digit selectors.

MESSAGE CHARACTERS

Message characters are non-numeric characters, which will print as specified in the picture template. Picture templates can contain a separator character for the fractional part of a number, and also a three-digit separator character (or thousands separator). By default, these characters are set to a decimal point and a comma respectively, as in the following picture format:



It is important to know that when using digit selectors in a picture template, the first character *must always* be a digit selector. Any non-numeric characters specified before the first digit selector will be ignored and will not appear in the formatted output. To overcome this problem and include characters before the first digit selector, the PREFIX= option must be used. Therefore, to include a pound sign in the bank format, we cannot merely put a '£' at the front of the template. Instead we must use the PREFIX='£' option:

```
proc format;
  picture bank
    low-<0  = '000,000,009.99  D' (prefix='£')
    0       = '9.99'              (prefix='£')
    0<-high = '000,000,009.99 CR' (prefix='£');
run;
```

Note that the PREFIX= option only applies to the range immediately preceding it, as opposed to the whole format. Therefore to include a pound sign in front of all formatted values produced by the bank format, the PREFIX= option must be included 3 times, once for each range.

DATE AND TIME DIRECTIVES

Directives are special characters that can be used in picture templates for formatting date, time and datetime values. There are 11 directives for formatting date values:

Directive	Output Generated	Possible Output
%a	Abbreviated weekday name	e.g. Fri
%A	Full weekday name	e.g. Friday
%b	Abbreviated month name	e.g. APR
%B	Full month name	e.g. April
%d	Day of month as decimal number	1 – 31
%j	Day of year as decimal number	1 – 366
%m	Month as decimal number	1 – 12
%U	Week number of the year as decimal (Sunday used as 1st day of week)	0 – 53
%w	Weekday as a decimal number	1 – 7 (1=Sunday)

%y	Year as decimal number, without century	0 – 99
%Y	Year as decimal number, with century	e.g. 2004

```
proc format;
  picture DateA other ='%d-%b-%y'(datatype=date);
  picture DateB other ='%A-%B-%Y'(datatype=date);
  picture DateC other ='%d*%m*%y'(datatype=date);
run;
```

April fool's day this year ('01APR2004'd, SAS date value 16162) is printed as 1-APR-4 when using the DATEA format, as Thursday-April-2004 when using the DATEB format, and as 1*4*4 using the DATEC format.
You will notice that by default, the directives that produce numbers do not print a leading zero (e.g. 1-APR-4 produced using DATEA format above). If a leading zero is required then just add 0 (zero) before the directive. For example, to insist on printing leading zeros in the date formats above, the DATEA and DATEC formats could be re-written as follows:

```
proc format;
  picture DateA other ='%0d-%b-%0y'(datatype=date);
  picture DateC other ='%0d*%0m*%0y'(datatype=date);
run;
```

Now April fool's day will be printed as 01-APR-04 when using the DATEA format, and as 01*04*04 using the DATEC format.

There are 5 directives for formatting time values:

Directive	Output Generated	Possible Output
%H	Hour as decimal number (24hr clock)	1 – 24
%I	Hour as decimal number (12hr clock)	1 – 12
%M	Minute as decimal number	0 – 59
%p	Morning or Afternoon; AM or PM	AM or PM
%S	Second as decimal number	0 – 59

```
proc format;
  picture TimeA other = '%H (h) %M (m) %S (s)'
          (datatype=TIME);

  picture TimeB other = 'The time is %I:%M %p'
          (datatype=TIME);
run;
```

The time at one second to midnight ('23:59:59't, SAS time value 86399) is printed as "23 (h) 59 (m) 59 (s)" when using the TIMEA format and as "The time is 11:59 PM" when using the TIMEB format.

Below are some further things to be aware of when using these directives:

- The directives must be used in conjunction with the appropriate DATATYPE= option (set to DATE, TIME or DATETIME accordingly). This option goes in parentheses after the picture template
- The directives are case sensitive so take care when typing them
- Be sure to use single quotes for picture templates when using directives. If you use double quotes, the % symbol followed by a non-blank character will cause the macro facility to interpret the directives as macro calls

We can sometimes run into problems when using directives if we need to use a percent sign (%) within our picture template:

```
proc format;
  picture profits other='%profit for month %m'
          (datatype=DATETIME);
run;
```

Applying the profits format to the datetime '01APR04:23:59:59'dt (one second to midnight on April fool's day) will print as "PMrofit for month 4". However I was actually hoping for it to print as "%profit for month 4". In this example, the %p in the template has been interpreted as the AM / PM time directive, hence explaining why "PMrofit" was printed in the formatted output. To overcome this problem we can use the %% directive in our picture templates to stop SAS from interpreting the % as part of a directive. The corrected format would be written as follows:

```
proc format;
  picture profits other='%%profit for month %m'
          (datatype=DATETIME);
run;
```

USEFUL PICTURE FORMAT OPTIONS

DIG3SEP= AND DECSEP= OPTIONS

The DIG3SEP= option can be used to change the character used as the three-digit separator or thousands separator, which by default is represented by a comma. Similarly the DECSEP= option will change the character used as the fractional separator, by default set as a decimal point.

```
proc format;
  picture euros other = '000.000,00'
    (prefix='EUR' decsep=',' dig3sep='.');
run;
```

The EUROS format applied to the number 123456.78 would print EUR 123.456,78

FILL= OPTION

When there are not enough digits in the variable value to completely fill a picture template, the character used to fill in the gaps varies depending on the type of digit selector in the unfilled position. By default, a conditional digit selector (a zero) is filled with a blank, whereas an absolute digit selector (non-zero number) is filled with a zero. An alternative fill character for conditional digit selectors can be specified instead of the default blank by using the FILL= option:

```
proc format;
  picture fillchar other = '000,000.00'
    (prefix='£' fill='*');
run;
```

Numerical value	Formatted value with FILLCHAR format
1	*****£1.00
12	****£12.00
123	***£123.00
1234	*£1,234.00
12345	£12,345.00

- If the FILL= option is used in conjunction with the PREFIX= option, then the prefix character(s) is added before the fill character completes the unfilled positions
- Notice how the formatted values above all display 2 decimal places even though conditional digit selectors were used in the picture template: Don't forget that any digit selectors included after the decimal point always behave as absolute digit selectors
- If there are any other characters in your template,

such as a comma, which appear to the left of the last filled digit selector position, then these characters are replaced by the fill character (conditional digit selectors) or leading zero (absolute digit selectors)

- An absolute digit selector will always print a zero if the position is unfilled. In other words the FILL= option has no effect on absolute digit selectors

MULT= AND ROUND OPTIONS

The following picture format attempts to display large numbers in millions. For example, I am aiming to format the number 9,876,543.21 as 9.88 million.

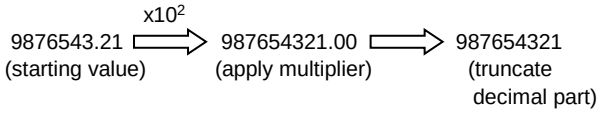
```
proc format;
  picture MillionA other = '9.99 million';
run;
```

Unfortunately, applying the MILLIONA format to the number 9876543.21 actually prints "3.21 million". To realise why this has happened we need to understand how numerical values are placed into our picture templates.

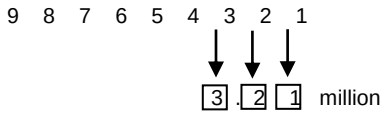
First of all we need to understand the role of multipliers in picture formats. By default, the numerical value being formatted is first multiplied by 10^n, where n is the number of digits after the decimal point (or fractional separator) in the picture template. Now in our MILLIONA format there are two digits after the decimal point in the picture template. This will mean that any numerical values being formatted by MILLIONA are first multiplied by 10^2.

$9876543.21 * 10^2 = 9876543.21 * 100 = 987654321.00$

Secondly, we need to know how numerical values are rounded or truncated before formatting. By default, the format first multiplies the variable value by the multiplier, and then truncates the decimal part (if any).



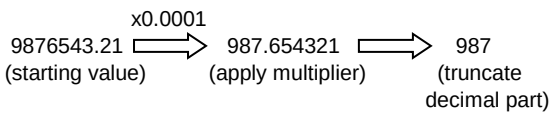
Once the numerical value has been multiplied and truncated, it is then placed into the picture template. The numbers are placed into the template starting with the rightmost character.



In our case, the picture template only contains 3 digit selectors, and so is not wide enough to accommodate our 9 digit number. Therefore only the 3 rightmost digits in our number are printed by the MILLIONA format, hence the formatted value appears as "3.21 million".

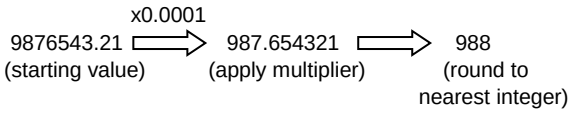
To get the format working as originally intended, we need to be able to specify an alternative multiplier to be used instead of the default 10^n. This can be done by using the MULT= option. The multiplier we need to use in this case is 0.0001.

```
proc format;
  picture MillionB
    other = '9.99 million' (mult=0.0001);
run;
```

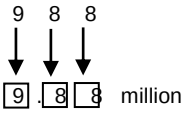


Now when our multiplied and truncated value is placed within our picture template we can see that the formatted value will be "9.87 million". However this isn't quite what I was hoping for. I actually wanted the number to be rounded to the nearest 0.01 of a million (i.e. 9.88 million instead of 9.87 million). To get the format to do this, all we have to do is add the ROUND option. Then instead of just truncating the decimal part, the ROUND option will round the value to the nearest integer before formatting.

```
proc format;
  picture MillionC (round)
    other = '9.99 million' (mult=0.0001);
run;
```



Now we can see that the number 9876543.21 will be printed as "9.88 million" when formatted with the MILLIONC format.



NOEDIT OPTION

In the picture format that follows I am intending to format any teenagers as "13+ years old":

```
proc format;
  picture AgeA
    1       = '1 Year Old'
    2 - 12  = '09 Years Old'
    13-high = '13+ Years Old';
run;
```

However, don't forget that any non-zero digit placed in a picture template will be interpreted as an absolute placeholder. Don't fall into the trap that only the number 9 fulfils this role. Therefore, the number 17, when formatted with the AGEA format will actually print as "17+ Years Old". To force SAS to interpret numbers as message characters instead of digit selectors we can use the NOEDIT option in parentheses after the value in question:

```
proc format;
  picture AgeB
    1       = '1 Year Old'
    2 - 12  = '09 Years Old'
    13-high = '13+ Years Old' (noedit);
run;
```

Now the number 17, when formatted with the AGEB format will print as "13+ Years Old" as originally intended.

NESTED FORMATS

Let's suppose we need to read some date values into a SAS date variable. Let's further suppose that the majority of the date values are in DATE9. format (e.g. 25DEC2004), however some values may be either "Unknown" or "N/A". How could we go about reading these dates into a SAS date variable?

One approach could be to read all the date values in using the SAS DATE9. informat, ignoring the invalid data messages that will appear in the log. Any "Unknown" or "N/A" values will

simply be assigned a missing value and we will no longer be able to distinguish between these two different values.

A far better approach would be to define our own informat to read in “Unknown” and “N/A” as special missing values, enabling these values to still be differentiable. We can then leave the actual dates to be read in with the SAS DATE9. informat as before. This could be achieved using the following nested informat:

```
proc format;
  invalue dates
    "Unknown"    = .U
    "N/A"        = .N
    other        = [Date9.];
run;
```

A nested informat is an informat where one or more of the values is another existing informat (either a SAS informat or another user-defined informat). The principle of nested formats is the same.
The screen shots below show our new nested informat, DATES, in action:

```
data read_dates_in;
input SAS_Date dates.;
Formatted_Date=put(SAS_Date,weekdate.);
if missing(SAS_Date) then Missing='Yes';
else Missing='No';
cards;
31DEC1959
Unknown
02JAN1960
N/A
run;
```

Output - Print of read_dates_in Data Set.lst

SAS_Date	Formatted_Date	Missing
-1	Thursday, December 31, 1959	No
U	U	Yes
1	Saturday, January 2, 1960	No
N	N	Yes

Notice how the SAS_date variable is indeed missing for the values that were originally “Unknown” and “N/A”, but these two missing values remain distinguishable.

Moving on, I would now like to create a format to divide the integers from 1 to 20 into three different groups:

- Prime Numbers
- Under 10 and non prime
- 10 and above, and non prime

(A prime number is an integer greater than one whose only positive divisors (factors) are one and itself)

A first attempt at creating this format could look something like the following:

```
proc format;
  value PrimeA
    2,3,5,7,11,13,17,19 = 'Prime Number'
    low - < 10          = 'Under 10 & Non Prime'
    10  - high          = '10 plus & Non Prime';
run;
```

Unfortunately this generates errors when run, due to overlapping ranges, as shown in the log:

```
165      proc format;
166        value PrimeA
167          2,3,5,7,11,13,17,19 = 'Prime Number'
168          low - < 10          = 'Under 10 & Non Prime'
169          10  - high          = '10 plus & Non Prime';
ERROR: These two ranges overlap: LOW-<10 and 2-2 (fuzz=1E-12).
ERROR: These two ranges overlap: 10-HIGH and 11-11 (fuzz=1E-12).
NOTE: The previous statement has been deleted.
170      run;
```

To get around this problem, we can create a nested format:

```
proc format;
  value PrimeB
    2,3,5,7,11,13,17,19 = 'Prime Number'
    other               = [PrimeC.];

  value PrimeC
    low - < 10  = 'Under 10 & Non Prime'
    10  - high  = '10 plus & Non Prime';
run;
```

The PRIMEC format has been nested within the PRIMEB format. This will mean that the non prime numbers are subsetted further by applying the PRIMEC format to them. I can now obtain the desired output by formatting my numbers with the PRIMEB format.

```
data Numbers;
  do Number=1 to 20;
    Formatted=put(number,PrimeB.);
    output;
  end;
run;
```

Output - Print of the Numbers Data Set.lst

Number	Formatted
1	Under 10 & Non Prime
2	Prime Number
3	Prime Number
4	Under 10 & Non Prime
5	Prime Number
6	Under 10 & Non Prime
7	Prime Number
8	Under 10 & Non Prime
9	Under 10 & Non Prime
10	10 plus & Non Prime
11	Prime Number
12	10 plus & Non Prime
13	Prime Number
14	10 plus & Non Prime
15	10 plus & Non Prime
16	10 plus & Non Prime
17	Prime Number
18	10 plus & Non Prime
19	Prime Number
20	10 plus & Non Prime

- We have see that nested (in)formats have two useful applications:
  - Handling values that are known to be outside the expected or accepted range of values (as in the first example where we created the dates informat to deal with unacceptable DATE9. values)
  - Subsetting values out of larger ranges (as in the second example where we created the PRIMEB format to subset the non prime numbers)
- Be sure to use square brackets [ ] around nested format names. If you prefer you can use parentheses and vertical bars (|…|). If you forget these brackets then default quoting will occur and the actual format name will become the formatted value
- Don’t forget the period at the end of the nested format name (e.g. [PrimeC.] in our example)

- Try to avoid creating nested formats with more than one level since these will dramatically increase resource requirements
- You cannot use nested formats with the PRELOADFMT option in the TABULATE, MEANS or SUMMARY procedures

MULTILABEL FORMATS

When using SAS Version 8 or higher, the MULTILABEL option enables you to define formats where ranges can be formatted as more than one value. That is, you are able to assign multiple values to overlapping ranges.

For example, consider some data containing exam results for students. The following multilabel format summarises the exam results in two different ways:

```
proc format;
  value scores (multilabel)
    low-25       = '25% and below'
    25<-50       = '25% to 50%'
    50<-75       = '50% to 75%'
    75<-high     = 'Above 75%'

    70-high      = 'Grade A'
    60-<70       = 'Grade B'
    50-<60       = 'Grade C'
    40-<50       = 'Grade D'
    low-<40      = 'Ungraded';
run;
```

The MEANS, SUMMARY and TABULATE procedures can use these multilabel formats to powerful effect, enabling you to simultaneously summarise your data in multiple groupings. To make use of a multilabel format in these procedures you need to use the MLF option on the class statement:

```
proc tabulate data=exam_results;
  format score scores.;
  class score gender / mlf;
  table score='' all,
        gender='Gender'*n*f=8. all*n*f=8.
        /box='Exam Results';
  keylabel n=' ' all='Total';
run;
```

Exam Results	Gender		Total
	Female	Male	
25% and below	2	1	3
25% to 50%	16	28	44
50% to 75%	12	40	52
Above 75%	2	3	5
Grade A	5	9	14
Grade B	5	15	20
Grade C	4	19	23
Grade D	9	6	15
Ungraded	9	23	32
Total	32	72	104

SUM 104

SUM 104

If you use a multilabel format in any other procedure or in the data step, then only the primary label will be recognised. The primary label is the value that is assigned to the first internal value or range of internal values that matches or contains the entry when all internal values are ordered sequentially. For example, in our scores format, the primary label for 55 is “Grade C” because the range 50-<60 is sequentially the first range of internal values that contains 55. The secondary label for 55 is “50% to 75%” because the range 50<-75 occurs in sequence after the range 50-<60.

Take care when using the NOTSORTED and MULTILABEL options together. Combining them is often unreliable.

STORING FORMAT DEFINITIONS IN DATASETS

CREATE A FORMAT FROM A DATASET

So far we have primarily focussed on using the VALUE, INVALUE and PICTURE statements of PROC FORMAT to create our own user-defined formats. It is possible however to create formats without having to use these statements. In fact, we can create a format from the variables contained within a data set by specifying a so-called input control data set with the CNTLIN= option of PROC FORMAT.

The input control data set is a regular SAS data set that must contain variables of specific names which contain the necessary information to define a format. Each row of the CNTLIN data set contains the equivalent of a PROC FORMAT range-value pair.

Your CNTLIN data set must always contain the following three variables:

- FMTNAME
  This character variable must contain the name of the format. The name must be the same for all rows that make up the format

- START
  This variable contains the starting values of the ranges

- LABEL
  This variable contains the formatted or informatted values

As a very simple example, the following data set could be used to create a format for a gender variable containing values M and F:

VIEWTABLE: Work.Gender_input_control_data_set

	FMTNAME	START	LABEL
1	$gender	M	Male
2	$gender	F	Female

The PROC FORMAT code to create the $GENDER format is now as straightforward as:

```
proc format cntlin=gender_input_control_data_set;
run;

* which is equivalent to the following
  standard value statement approach... ;
proc format;
  value $gender 'M'='Male'
                'F'='Female';
run;
```

Going beyond the three compulsory variables, it is likely that you will need to know some of the other optional CNTLIN data set variables. The following variables can be used for defining the specifics of the format ranges:

- END

  This variable contains the end values of the ranges. If this variable is omitted then the END is deemed to be the same as the START

- SEXCL and EEXCL

  These variables define whether or not the start and end values are to be excluded from the range. Their permissible values are:
  - **Y** (Yes - exclude from range)
  - **N** (No - include in range)

  The following table gives an example of their use:

DESIRED RANGE	SEXCL	EEXCL
5 – 10 (5 to 10 inclusive)	N	N
5 <– 10 (exclude 5, include 10)	Y	N
5 –< 10 (include 5, exclude 10)	N	Y
5 <–< 10 (5 to 10 exclusive)	Y	Y

- HLO

  This variable contains additional range information in the form of a string, made up from any combination of the following 8 characters:

F	standard SAS (in)format used for (in)formatted value
H	end value of range is HIGH
I	numeric informat range (used for unquoted numeric range)
L	start value of range is LOW
N	format has no ranges, not even OTHER
O	range is OTHER
M	MULTILABEL option to be used
R	ROUND option to be used (pictures only)
S	NOTSORTED option to be used

  For example, defining HLO='LH' will specify a low-high= range.

The following variables are general to the entire format rather than a specific range. Therefore, these variables must have a consistent value across all the rows used to make up the format.

- TYPE

  This variable defines the type of format being created. Permissible values for this variable:
  - **C** (Character format)
  - **I** (Numeric informat)
  - **J** (Character informat)
  - **N** (Numeric format excluding pictures)
  - **P** (Picture format)

- DEFAULT – Default length of format value
- FUZZ – The fuzz factor
- LENGTH – Length of the format
- MAX – Maximum length of format value
- MIN – Minimum length of format value

Finally, the following variables can be used to utilise the PICTURE format options that we covered earlier: DATATYPE, DECSEP, DIG3SEP, FILL, MULT, NOEDIT and PREFIX.

CREATE A DATASET FROM A FORMAT

Having seen how to create a format from a data set, we are now going to go the other direction and see how to create a data set from a format, with the help of the CNTLOUT= option. The following code will create an output control data set called WORK.MY_FORMATS to contain all the formats and informats stored in the WORK.FORMATS catalog:

```
proc format library=work.formats
            cntlout=work.my_formats;
run;
```

The new MY_FORMATS output control data set contains the following 21 columns, most of which should look familiar from the previous discussion of CNTLIN data set variables:

Column Name	Type	Length	Label
FMTNAME	Text	32	Format name
START	Text	16	Starting value for format
END	Text	16	Ending value for format
LABEL	Text	17	Format value label
MIN	Number	3	Minimum length
MAX	Number	3	Maximum length
DEFAULT	Number	3	Default length
LENGTH	Number	3	Format length
FUZZ	Number	8	Fuzz value
PREFIX	Text	2	Prefix characters
MULT	Number	8	Multiplier
FILL	Text	1	Fill character
NOEDIT	Number	3	Is picture string noedit?
TYPE	Text	1	Type of format
SEXCL	Text	1	Start exclusion
EEXCL	Text	1	End exclusion
HLO	Text	11	Additional information
DECSEP	Text	1	Decimal separator
DIG3SEP	Text	1	Three-digit separator
DATATYPE	Text	8	Date/time/datetime?
LANGUAGE	Text	8	Language for date strings

The output control data set has one row for every range-value pair from each format or informat stored in the catalog specified in the LIBRARY= option. Output control data sets have several useful applications:

- Once the format definition is in a data set you can use data step programming to edit or manipulate the format
- Formats stored in data sets can be subsetted within a data step to create new formats or informats
- Once you have edited the data set it can be used as an input control data set in a subsequent PROC FORMAT step with the use of the CNTLIN= option

If you don't want to unload all the formats from the format catalog into your output control data set, then you can selectively choose the ones you do want by using the SELECT and EXCLUDE statements as mentioned earlier when we covered the FMTLIB option.

CONCLUSION

When uncovered and understood, PROC FORMAT is a very powerful Base SAS procedure. We have seen that there is a lot more to PROC FORMAT than just the value statement. I hope I have given you some new ideas for enhancing your programs, using advanced features of the format procedure.

ACKNOWLEDGMENTS

"More than Just Value: A look into the depths of PROC FORMAT"
Paper written by Peter J. Lund
http://www2.sas.com/proceedings/sugi26/front/toc.pdf

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact Amadeus at:

Company: Amadeus Software Limited
Address: The Old School Hall, 11 Wesley Walk
Witney, Oxon OX28 6ZJ
Work Phone: +44 (0) 1993 848010
Email: info@amadeus.co.uk
Web: www.amadeus.co.uk