Paper 4-27

More than Just Value: A Look Into the Depths of PROC FORMAT

Pete Lund, Northwest Crime and Social Research, Olympia, WA

Abstract

It doesn't take long for even novice SAS® programmers to get their feet wet with PROC FORMAT. It usually consists of a VALUE statement to recode a race or gender variable, or assign a state name to an abbreviation. It might have even been real fancy and used ranges to collapse ages into groups. All too often that's as far as the use of PROC FORMAT goes, which is a shame because there's a lot of power at your fingertips if you dig a little deeper.

This paper covers some of the lesser-used aspects of PROC FORMAT, including the PICTURE statement, CNTLIN/CNTLOUT datasets, nested formats, multi-level (overlapping range) formats, and permanent formats and the FMTSEARCH option. There are some things that are new to Version 8, others that just seem to be some of those undiscovered gems, and a few "gotchas" to be aware of. A basic knowledge of PROC FORMAT and VALUE/INVALUE statements are assumed.

Introduction

As is true with many aspects of programming, when creating your own formats or informats it is imperative to "KNOW YOUR DATA!" You can avoid a number of the "gotchas" and take advantage of a number of the features of formats that will be discussed in this paper if you have a good feel for the data to which they will be applied.

For purposes of readability, I'll use the term "format" as generic for both formats and informats, distinguishing between the two when necessary.

PROC FORMAT Notes and Options

It usually isn't long into the journey of SAS programming before one encounters the need to use PROC FORMAT. There are two major reasons to use SAS formats: 1) labeling values and 2) grouping values. Most SAS programmers are familiar with the VALUE and INVALUE statements that accomplish these tasks. Even more "esoteric" uses of formats, like table lookups and range validation, can be fit into one of these two uses.

There are a few general notes about formats and format options that I'll cover before we get into some more specific topics.

Format Names

There can be a "gotcha" when naming your formats. Format names can be no longer than eight characters, even in Version 8. This includes the $ sign on character formats. Informat names can also be up to eight characters long, which also includes the $ on character informats. However, keep in mind that there is an implied @ on the front of all informat names. So, numeric informats can really be seven characters long and character informats can really be six characters long.

Note: PROC FORMAT will not generate an error if the name is too long. It simply issues a note and creates the format/informat with a truncated name. For example,

```
invalue $MyInfmt
  '1' = 'A'
  '2' = 'B';
```

generates the following note in the log (notice the @ at the beginning of the name):

```
NOTE: The informat name '@$MYINFMT' exceeds 8
characters. Only the first 8 characters will
be used.
NOTE: Informat $MYINFM has been output.
```

When you try and use this format, in a datastep or procedure, the format name will be truncated there are well, this time with a warning:

```
WARNING 504-185: The informat name '@$MYINFMT'
exceeds 8 characters, and will be truncated.
```

The real danger with this is if you have multiple format names that are not unique through eight characters. Subsequent formats will overwrite previous formats, with only a note in the log.

Both of the following formats actually create formats of the same name, TOOLONGA.

```
value TooLongAName
  1 = ‘Group A’
  2 = ‘Group B’;
value TooLongAValue
  1 = ‘Fred’
  2 = ‘Wilma’;
```

Obviously, unless you were paying attention to your log you could be in for a big surprise when analyzing your results. Both TooLongAName and TooLongAValue will be truncated to TooLongA in both the PROC FORMAT where they are created and in any subsequent use. So, the values in TooLongAValue are the ones that will always be used.

More on Format Names and FMTLIB

The FMTLIB (or PAGE) option prints the values of a format to the output window. By default, all the formats/informats in the specified library are printed. The SELECT and EXCLUDE statements can narrow the formats that will be printed.

Suppose that you had character and numeric formats and character and numeric informats all named MyFmt. (This is possible, but for clarity/sanity purposes, I wouldn't recommend it.) These would be referenced in a SELECT or EXCLUDE as follows:

- Character format: $MyFmt
- Numeric format: MyFmt
- Character informat: @$MyFmt
- Numeric informat: @MyFmt

This is the only place that you'll need to know that the informats begin with a @ - but as mentioned above, knowing that the @ is really there will help you avoid giving your informats names that are inadvertently too long.

Default Quoting of Ranges and Values

A format accepts either character or numeric input and will always produce character output. The "$" on a character format denotes that it expects character <u>input</u>. On the other hand, an informat always expects character input and will produce either character or numeric output. The "$" on a character informat denotes that it will produce character <u>output</u>.

SAS uses this information to quote the appropriate ranges and/or values, even if you don't in your PROC FORMAT code. Default quoting takes place as follows:

- Character format: ranges and values
- Numeric format: values
- Character informat: ranges and values
- Numeric informat: ranges

For example, the following two formats are exactly the same:

```
value $TestA    /*values and ranges quouted */
    ‘1’ = ‘A’   /* as expected.             */
    ‘2’ = ‘B’
    ‘3’ = ‘C’;

value $TestB    /* values and ranges will be*/
    1 = A       /* quoted by default.       */
    2 = B
    3 = C;
```

For the sake of clarity I would suggest always explicitly quoting your formats, but be aware that SAS will do it for you if you don't.

"Overlapping" Range Begin/End Values

If the range end values "touch" each other, SAS will assign the value associated with the first range to the variable value. For example, the following format is legal syntax.

```
value Scores
  0 - 30 = ‘Range 1’
  30 - 60 = ‘Range 2’
  60 - 100 = ‘Range 3’;
```

However, it may not be clear that the value 30 would be assigned to 'Range 1' and 60 would be assigned to 'Range 2'. A better way to code this would be to use the "<" range option to eliminate the range begin or end value from the range. The following format could be recoded:

```
value Scores
  0 - 30 = ‘Range 1’
  30 <- 60 = ‘Range 2’
  60 <- 100 = ‘Range 3’;
```

The "<-" eliminates the beginning range value. So, Range 2 is defined as "30 to 60, not including 30." You can also use "-<" to eliminate the end range value or "<-<" to eliminate both values from the range.

The FUZZ= Option

The FUZZ= option can be used on the VALUE or INVALUE statement to specify a factor for matching values to the range. In the following example values from .8 – 1.2 would be assigned “A,” values from 1.8 – 2.2 would be assigned “B” and so on.

```
value test (fuzz=.2)
  1 = ‘A’
  2 = ‘B’
  3 = ‘C’;
```

However, be wary of specifying a FUZZ value that is as large (or larger) than one-half the gap between ranges. For example, consider the following format:

```
value test (fuzz=.6)
  1 = ‘A’
  2 = ‘B’
  3 = ‘C’;
```

If we “expand” the format with the fuzz value we have, in essence:

```
value test
  0.4 – 1.6 = ‘A’
  1.4 – 2.6 = ‘B’
  2.4 – 3.6 = ‘C’;
```

Notice that the ranges overlap and it is not always predictable which value will be assigned. PROC FORMAT does not issue an error or warning when the FUZZ option causes ranges to overlap.

The NOTSORTED Option

The NOTSORTED option on the VALUE or INVALUE statements cause the (in)format ranges to be stored in the order in which they are defined. By default, SAS will store the ranges in sorted order. This can have implications in two areas:

- Performance – if you know that certain values are more likely to occur in your data, place those ranges first in your (in)format to improve search time.
- Presentation – the format will be presented in defined order with the FMTLIB or PAGE. Also, in PROCs TABULATE, MEANS and SUMMARY the PRELOADFMT and ORDER=DATA options, used together, will cause the ranges to be displayed in defined order.

The JUST and UPCASE Options

There are a couple very useful options on the INVALUE statement that operate on the variable values before they are compared to the informat ranges.

The JUST option left-justifies the variable value before it is compared to the format ranges. This can be very useful if the length of the variable is longer than the length of the format range. For example,

```
invalue $Cat (just)
  ‘A’ = ‘Category 1’
  ‘B’ = ‘Category 2’;
```

If the string to which the informat $Cat. is applied is 3 characters long, using the JUST option would allow “ A”, “ A “, and “A “ to all be set to “Category 1.” Otherwise, only “A “ would be the only value to match.

The UPCASE option causes the variable value to be converted to upper case before it is compared to the range. In the following example the values “m” and “M” would both be set to “Male,” while “f” and “F” would both be set to “Female.”

```
invalue $Gender (upcase)
  ‘M’ = ‘Male’
  ‘F’ = ‘Female’;
```

The PICTURE Statement

The PICTURE statement has long been one of those much feared features of SAS, much like PROC TABULATE or SAS/GRAPH®. But, like TABULATE and GRAPH, once you master the concept and syntax of PICTURE it becomes a very useful and powerful tool.

A PICTURE is just a template for displaying numbers. The template consists of “digit selectors” which are placeholders for the digits in the variables value. There are two types of digit selectors:

- 0 – think of this as a “conditional” placeholder. If there is a digit for the location it will print, if not nothing will print.
- 9 – (or any non-zero digit) think of this as an “absolute” placeholder. If there is a digit for the location it will print, if not a “0” will print.

For example, with the value 35

- picture “0000” would print: 35

- picture “9999” would print: 0035

A PICTURE template can also contain a thousands separator, by default a comma, and a decimal separator (or fractional separator), by default a decimal point. As we will see, these two separators effect how the number is placed in the template. Additionally, text can be placed in the template that will display as written.

General PICTURE Notes

Just a couple notes before we start. These notes will be covered in more detail or in examples below.

- All text before the first digit selector is ignored. Use the PREFIX= option to add text to the beginning of the formatted value.
- Here’s a tongue-twister: All digit selectors following the first non-zero digit selector are treated as non-zero. For example, “9000” is the same as “9999”.
- All digits to the right of the decimal in the picture are displayed, regardless of the digit selector type. For example, “00.000” will always display three digits to the right of the decimal place, even though the default for a zero digit selector is a blank. The value 123.4 would display as “123.400” in that PICTURE.

PICTURE FORMAT IN FIVE EASY STEPS

Suppose you have data containing percentages that you want displayed with a percent sign, a comma thousands separator, a single decimal place and negative numbers surrounded by parentheses.

Let’s follow these values through the following PICTURE:

- -15.45 → (15.5%)
- 34.11 → 34.1%

```
picture RegPct (round)
  low -< 0  = ‘0,009.9%)’  (prefix=’(-‘)   ❶
   0 - high = ‘0,009.9%’;                   ❷
```

1. Like any format, the first step is to find the range into which the values fall. In our sample data, **-15.45** falls into range ❶ and **34.11** falls into range ❷.
2. Take the absolute value of the variable values (**15.45** and **34.11**).
3. Each range has either an explicit (with the MULT= option) or implicit multiplier. The implicit multiplier is 10^n, where n is the number of digits to the right of the decimal place in the template. In both our ranges, we have an implicit multiplier of 10 (10^1, single digit to the right of the decimal place). Apply the multiplier to the values and, because the ROUND option is specified, round the result to the nearest integer (**155** and **341**). By default, the values are truncated to the integer portion (154 and 341).
4. Fit the value into the template, beginning with the rightmost character:
   - **1 5 5**
   - □,□□□.□%)
   - **3 4 1**
   - □,□□□.□%
5. Apply prefix, if any. **(-15.5%)** and **34.1%**.

Note: if there are more digits in the variable value than there are digit selectors in the PICTURE, the rightmost digits will display. For example, with the above picture, the value 12345.6 would display as 2,345.6. Again, as noted earlier, know your data when you create your own formats.

PICTURE OPTIONS BY EXAMPLE

The best way to show some of the options for the PICTURE statement is with a few examples.

NOEDIT Option

```
picture ChgGrp
  1        = '1 Charge'
  2-14     = '09 Charges'
  15-high  = '15+ Charges'  (noedit);
```

As noted above, any digit in a PICTURE is a placeholder for a digit in the variable value. In the above PICTURE we want the range of values 15 and above to have the value “15+….” In order to keep the digits “1” and “5” from being treated as digit selectors, we use the NOEDIT option on that range. Without that option, a value of 27 would be displayed as “27+ Charges.”

MULT= Option

```
picture Billion
  0 - high = '009.99B' (mult=.0000001);
```

By default the variable value is multiplied by 10^n, where n is the number of digit selectors to the right of the first decimal point in the picture, before the picture is applied. This default can be overridden with the MULT= option.

In the above case, variables will be multiplied by .0000001, so as to display values in billions. For example, 1234567890 x .0000001 = 123 (after automatic truncation). When fit into the picture, the displayed value would be “1.23B.”

FILL= Option

If there are not enough digits in the variable value to fill the picture, the fill character depends on the digit selectors in the picture that are not filled. Unfilled zero digit selectors are filled with blanks. Unfilled non-zero digit selectors are filled with zeros. The fill character can also be specified with the FILL= option.

```
 0 – high = ‘0000’;                 ❶
 0 – high = ‘9999’;                 ❷
 0 – high = ‘0000’ (fill=’*’);      ❸
```

In the above PICTURES the value 123 would display as:

```
❶    123
❷   0123
❸   *123
```

DIG3SEP and DECSEP Options

You can use these options to change the default thousands separator, from a comma, and the default fractional separator, from a decimal point. The DIG3SEP= option specifies the thousands separator and the DECSEP= option specifies the fractional separator.

```
picture PictA
  0 - high = '0.000,00'
    (decsep=’,’ dig3sep=’.’);
```

Calling the above picture with the value 1234.56 would display 1.234,56.

Keep in mind that the default multiplier will now be based on the number of digit selectors to the right of the character specified in the DECSEP= option.

Note: The DECSEP= option really does not do anything. Simply placing a character, in this case a period, in the proper position in the template will suffice.

DATE/TIME DIRECTIVES

Until now the PICTURE statement has been limited to use with numeric variables, providing a template for displaying numbers. An exciting new feature available in Version 8 is a set of “directives” for formatting date, time and datetime variables.

There is a long list of directives for formatting every part of a date or time value: hour, minute, second, day, month, week and year. (See the Version 8 Procedures Guide, pg. 443, for the complete list.) Let’s take a look at those related to formatting dates.

- %a – displays weekday abbreviation
- %A – displays weekday name
- %b – displays month abbreviation
- %B – displays month name
- %d – display day of month number
- %j – displays day of the year number
- %m – displays month number
- %U – displays week of year number
- %w – displays weekday number
- %y – displays year (no century)
- %Y – displays year with century

Notice that the values are case-sensitive and be aware that the directives that produce a number (d, j, m, U, y) do not return leading zeros by default. We’ll see how to get them in the following examples.

There are two easy steps in using the PICTURE directives:

- Place them in a quoted string. More than one directive can be used in a given value. Note: Make sure to use single quotes!! Since the directives begin with % placing them inside double quotes will cause the macro compiler to try and interpret them as macro names.
- Tell PROC FORMAT that the value is using directives with the DATATYPE= option. Note: There is an error in the documentation concerning the DATATYPE= option. It states that it is a PICTURE statement option when it is, in fact, a range option. The valid values for DATATYPE= are:
    - DATE
    - TIME
    - DATETIME

Let’s look at a few examples using the SAS date value 15445 (4/15/2002).

```
picture  SUGI
  low-‘13apr02’d = ‘SUGI hasn’t started’
  ‘14apr02’d-‘17apr02’d = ‘It is %A of SUGI’
        (datatype=date)
  other = ‘Sorry, SUGI is over’;
```

Notice that the middle range contains the date directive %A which prints the day name. Calling this format with our test date would produce:

```
It is Monday of SUGI
```

There are two more things to notice about this PICTURE format:

- Not all the ranges are required to have a directive (and thus the DATATYPE= option)
- The directives can be placed in the midst of other text. Remember that a standard PICTURE must begin with a digit selector and any preceding text must be added with a PREFIX= option.

It was noted earlier that more than one directive can be used in a single value. For example,

```
picture XXX
  low - high = 'It is %B or %b or %m or %0m  '
      (datatype=date);
```

returns the following:

```
It is April or APR or 4 or 04
```

This example also demonstrates how to get a leading 0 on directives that produce a number. The %m directive displays the month number. If we want January-September to display as 01-09, simply prefix the directive letter with a 0 – in this case, %0m. This same syntax works with all the directives that produces numbers.

Note: There is a “gotcha” when using the PICTURE directives. As with other formatted values created with PROC FORMAT the length of a value created by calling a PICTURE format is determined by the length of the longest formatted value. However, that length is determined by the length of the quoted string in the PICTURE statement, not by the length of the “expanded” values. For example, the following

```
picture MonYear
  low – high = ‘%B %Y’ (datatype=date);
```

would yield the following, probably undesired, result with our example date:

```
April
```

Notice that the format string “%B %Y” is 5 bytes long and the result contains only the first 5 characters of the desired result. Adding a MAX=13 or DEFAULT=13 option to the PICTURE statement will give the expected result.

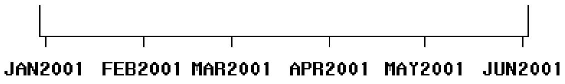
```
picture MonYr (default=13) /* or (max=13) */
  low – high = ‘%B %Y’ (datatype=date);
```

```
April 2002
```

The New Directives and SAS/GRAPH

It’s always nice when a feature of one part of SAS has a positive impact on another feature. We can take advantage of the new PICTURE directives in combination with the new SPLIT= option on the AXIS statement to achieve some nicer looking axis labels in SAS/GRAPH procedures.

Let’s suppose we were producing a plot with values from January through June 2001. A typical axis, using a MONYY7. format, would look like this:



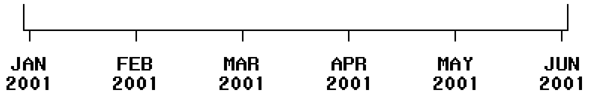
The axis values are starting to get a little crowded. We can take advantage of the PICTURE directives and the AXIS SPLIT= option to get a little more white space.

```
proc format;
  picture MonYr (default=8)
  low – high = ‘%b-%Y’ (datatype=date);
run;

axis1 split=’-‘;

proc gplot data=MyData;
  plot SomeData*TheMonth /
     haxis=axis1;
  format TheMonth MonYr.;
run;
```

Values for TheMonth will now look similar to the MONYY7. format, except that there will be a hyphen between the month abbreviation and the year (i.e., APR-2001). The SPLIT=’-‘ will break the axis label into multiple lines on the hyphen and we’ll get an axis like this:



We can now fit more month labels on the axis without a crowded look or resorting to smaller and smaller fonts.

Nested Formats

A “nested” format is simply one where one or more of the labels is another SAS format. It can be either a SAS-supplied or user-defined format.

One use for nested formats is to subset values out of larger ranges. In my work with jail data from King County, WA I often want to subset a few cities into one group and group the rest of the data into King County and the “rest of the state.” In the data, the first two digits of the city field are the county (King County = “17”). The following format would cause an error,

```
value $KCbreak
  ‘17001’,’17005’,’17011’  = ‘Urban Cities’
  ‘17000’ – ‘17999’        = ‘King County’
  other                    = ‘Rest of State’;
```

because the ranges overlap.

However, I could use a nested format to achieve the desired results.

```
value $KCbreak
    ‘17001’,’17005’,’17011’  = ‘Urban Cities’
    other                    = [$KCother.];
value $KCother
    ‘17000’ – ‘17999’        = ‘King County’
    other                    = ‘Rest of State’;
```

Now, if I call $KCbreak. with a value outside the three cities I’m interested in, the $KCother. format is called. Here are a few examples:

- 17001 → “Urban Cities”
- 17002 → [$KCother.] → “King County”
- 18001 → [$KCother.] → “Rest of State”

Another use of nested formats is for dealing with values that are known to be outside the expected range of valid values. For example, suppose that you have survey data which has a date-of-birth field. Two non-date values are used for different types of missing values; 888888 for “refused to answer” or 999999 for “unknown.” Obviously, these values are not valid dates, so you have three choices when reading this file:

- use a MMDDYY. informat and ignore the error messages
- read the variable as a string and test for the invalid values and then use an INPUT function to convert the valid values to SAS dates
- use a nested informat that tests for the “bad” values and sets them to missing and nests the MMDDYY format for the other values

The nested informat is shown below:

```
invalue SetDate
  888888 = .R
  999999 = .U
  other = [mmddyy.];
```

Here, 888888 and 999999 will be set to missing (still separately identifiable) and the rest of the values are subject to the mmddyy. informat.

To “nest” a format in SAS for Windows, simply place the other format name in brackets, or parentheses and vertical bars (|…|). Note: The nesting symbol may be different on different operating systems. See your OS guide for details.

Note: There is a “gotcha” when nesting formats. Be sure to include the brackets around the nested format name! It was stated earlier that the quotes on a format label are implied. If you use a format name without the brackets, SAS will assume that it is text and assign that format name rather than the associated formatted value.

Note: The PRELOADFMT option in PROC TABULATE, MEANS or SUMMARY cannot be used with nested formats.

MultiLabel Formats

The new, in version 8, MULTILABEL option allows you to assign more than one range to a single value. Certain SAS procedures (TABULATE, MEANS and SUMMARY) can take advantage of these formats and produce output that counts a single observation in more than one place.

For example, consider that we have school data with age of student. The following format breaks down the age in two ways.

```
value AgeGrp (multilabel)
  0 – 4   = ‘Under 5’
  5 – 9   = ‘5 – 9’
 10 – 14  = ’10 – 14’
 15 – 19  = ‘15+’
```

```
  0 - 12 = 'Pre-teen'
 13 - 19 = 'Teen';
```

To use this format in a procedure you specify the MLF option on the class statement. For example, to use the above format in a PROC MEANS:

```
proc means data=AgeData min max mean;
  class age / mlf;
  var height;
  format age agegrp.;
run;
```

Most procedures, and the PUT function, will look at the first set of ranges to determine the value to assign. This is called the primary format.

Note: You may want to experiment with the results of multilabel formats. At this time (through version 8.2) there is no reliable way to use both the NOTSORTED and MULTILABEL option together. This means that the ranges will all be ordered sequentially as though they were one format, not necessarily in the order they were arranged in the procedure.

CNTLIN Datasets

It is possible to build a format without coding PROC FORMAT statements. A dataset containing the information to create the format can be accessed using the CNTLIN= option.

The CNTLIN dataset is regular SAS dataset, containing variables of specific names that contain the format information. Each record contains the equivalent of a range-value pair in PROC FORMAT.

The basic variables are:

- FMTNAME – the name of the format, without any "$" or "@". This must have the same value on all records for the format.
- TYPE – specifies the type of the (in)format. The valid values are:
  - N – numeric format (the default)
  - C – character format
  - I – numeric informat
  - J – character informat
  - P – picture format
- START – the starting value of the range.
- END – the ending value of the range. If this variable is not present, the END is set equal to the START for each record.
- VALUE – the value to be associated with the START-END range.

Of these, FMTNAME, START and VALUE are required. There are a few other variables that can help further specify your ranges.

- SEXCL – a flag that (Y or N) that designates whether the start value should be excluded from the range. The default is "N' and this variable is not required. Reflects the sss <-eee = vvv range syntax.
- EEXCL - a flag that (Y or N) that designates whether the end value should be excluded from the range. The default is "N' and this variable is not required. Reflects the sss -<eee = vvv range syntax.

  Note: both SEXCL and EEXCL can be set to "Y" on the same range. This reflects the sss <-< eee = vvv range syntax and indicates that both the start and end values are excluded from the range.
- HLO – a string that can contain a mixture of the following values:
  - H – the end value is "HIGH." This option changes any specified END value to 'HIGH."
  - L – the start value is "LOW." This option changes any specified START value to "LOW."
  - O – the range value is "OTHER." This option changes any specified START and END values to "**OTHER**."
  - S – the NOTSORTED option is in effect. The NOTSORTED option is a format-level option. This value must be present on at least the first record in the CNTLIN dataset for it to be utilized.
  - R – the PICTURE statement ROUND option is in effect.
  - N – the format has no ranges, including no "OTHER" range.

  Note: The value of HLO can be a combined of the above values. For example, use HLO='LH' to specify a low-high=… range.

There are a number of CNTLIN variables that are general to the format rather than specific to a range. These variables should have the same value on every record in the dataset.

- MIN – the minimum length of the value
- MAX – the maximum length of the value

- LENGTH – the format length
- FUZZ – the fuzz factor
- DEFAULT – the default length of the value

In addition, there are CNTLIN variables for all the PICTURE options (see above discussion of PICTURE for details).

- DATATYPE
- DECSEP
- DIG3SEP
- FILL
- MULT
- NOEDIT
- PREFIX

Creating SAS Formats from External Files

It is often the case that source data, descriptive data or metadata that are used in our SAS processes are stored and maintained in another format. It could be a database, spreadsheet or flat files. With the capabilities that SAS provides we can leave that data where it resides and still take advantage of the descriptive and metadata to label, group and subset the SAS data that we use.

For example, let’s suppose you have an ODBC data source, called FacCode, pointing to a spreadsheet like the following, in a named range called FacDesc. This information is used by a number of applications and, for maintainability sake, we want to have that single source of descriptive data.

Facility Code	Description
101	Minimum Security
102	Medium Security
201	Electronic Home Detention
301	Work Release

This simple program would create a format mapping facility code to the description.

```
proc sql;
  connect to odbc(dsn=”FacCode”);
  create table Facs as
  select start,
         label,
         ‘C’ as type,
         ‘FacCode’ as fmtname
  from connection to odbc
    (select [Facility Code] as start,
            [Description] as label
     from FacDesc);
quit;

proc format cntlin=Facs;
run;
```

The SQL query will read the spreadsheet and create variable START from the Facility Code column and the variable LABEL from the Description column. The FMTNAME and TYPE variables are set to constants, giving the format name and type.

We now have a dataset, Facs, that looks like this:

FmtName	Type	Start	Label
FacCode	C	101	Minimum Security
FacCode	C	102	Medium Security
FacCode	C	201	Electronic Home Detention
FacCode	C	301	Work Release

Notice that the dataset contains the necessary variable names. When it is brought into PROC FORMAT with the CNTLIN= option a format, FacCode, will be created. This is equivalent to the following PROC FORMAT code:

```
value $FacCode
  ‘101’ = ‘Minimum Security’
  ‘102’ = ‘Medium Security’
  ‘201’ = ‘Electronic Home Detention’
  ‘301’ = ‘Work Release’;
```

We have the flexibility of keeping the descriptive data in one place that is easily maintainable and not having to change the SAS program that creates the format. If the spreadsheet changes and the SAS code rerun, the format will reflect the changes.

Getting More than Your Monies Worth

Another “trick” we can do with PROC FORMAT is to assign “multiple values” in a single format call. Suppose we had a dataset like the following that contain descriptive information about different crime codes. For each crime code we know whether it’s a misdemeanor or felony, have a severity score and a mapping to another classification scheme.

Crime	Class	Severity	KCJ
100011	Misd	1	52
100012	Misd	3	54
100015	Misd	1	45
100022	Felony	6	52
100026	Felony	8	55
100030	Misd	4	46

We could create three separate formats, one for each of the additional pieces of information. Alternatively, we could create a single format containing all the information and parse it out in our code.

Let’s assume that the above is in a SAS dataset. The following code will create the CNTLIN dataset, generate the format and then use that format to create three new variables in a dataset that has only the crime code value.

```
data crimes(keep=fmtname type
                 start label hlo);
  set aaa.CrimeCodes end=done;

  retain fmtname ‘CrimeCd’ type ‘C’;

  start = Crime;
  label = trim(class)||’~’||
          put(severity,1.)||’~’||
          kcj;
  output;

  if done then
    do;
      hlo = ‘O’;
      label = ‘UNK~.~00’;
      output;
    end;
run;

proc format cntlin=crimes;
run;
```

We simply put the three values together to create the label, separated by a tilde (or any character that is not part of any value). Notice that at the end of the file (end=done) we want to create an “Other” category. We can do this by setting the HLO flag to “O” and giving the label we want assigned to other values. It makes no difference what the START value is for this record. When PROC FORMAT sees the HLO=”O” value the START will automatically be changed to “**OTHER**.” The above code is the same as the following format:

```
value $CrimeCd
  ‘100011’ = ‘Misd~1~52’
  ‘100012’ = ‘Misd~3~54’
           :
  ‘100030’ = ‘Misd~4~46’
  other    = ‘UNK~.~.’;
```

When we use this format, the three pieces of information can be parsed out with the SCAN function:

```
data aaa.CrimesPlus(drop=CrimeInfo);
  set aaa.OnlyCrimes;

  CrimeInfo = put(CrimeCode,$CrimeCd.);
  Class = scan(CrimeInfo,1,’~’);
  Severity = input(scan(CrimeInfo,2,’~’),1.);
  KCJ = scan(CrimeInfo,3,’~’);
run;
```

Building Lookup Tables

One of the most common uses of building formats from datasets is creating a “lookup table.” In the following example we have a dataset, SAMPLE, which contains an ID field. We want to extract and analyze records from a MASTER dataset that match to the IDs in the SAMPLE.

```
data KeepIDs;
  set aaa.sample;

  retain fmtname ‘KeepID’ type ‘C’;

  start = ID;
  label = ‘Y’;
run;

proc format cntlin=KeepIDs;
run;
```

The above code will create a format that maps all the IDs in our SAMPLE dataset to a ‘Y.’ We can use this format to subset a MASTER dataset for analysis without having to create another dataset.

```
proc tabulate data=aaa.MASTER;
   class….;
   table….;
   where put(ID,$KeepID.) eq ‘Y’;
run;
```

Obviously, this format could be used to subset the MASTER dataset prior to the TABULATE, but we can eliminate the need to create an additional dataset by applying the lookup in the procedure itself.

AUTHOR CONTACT INFORMATION

Pete Lund
Northwest Crime and Social Research
215 Legion Way SW
Olympia, WA 98501
(360) 528-8970
pete.lund@nwcsr.com
www.nwcsr.com

TRADEMARK INFORMATION

SAS is a registered trademark of SAS Institute Inc., Cary, NC, USA.