Paper BB70

FORMATs Top Ten

Christianna Williams PhD, Chapel Hill, NC

ABSTRACT

SAS FORMATs can be used in so many different ways! Even the most basic FORMAT use of modifying the way a SAS data value is displayed (without changing the underlying data value) holds a variety of nifty tricks, such as nesting formats, formats that affect various style attributes (such as alignment, font etc.), and conditional formatting. Add in PICTURE formats, multi-label FORMATs, using FORMATs for data cleaning, and FORMATs for joins and table look-ups, and we have quite a bag of tricks for the humble SAS FORMAT and the PROC FORMAT used to generate them. The purpose of this paper is to describe a few handfuls of very useful programming techniques that employ SAS FORMATs. While this paper will be appropriate for the newest SAS user, it will also focus on some of the lesser-known features of FORMATs and PROC FORMAT and so should be useful for even quite experienced users of SAS.

INTRODUCTION

Many new SAS programmers, particularly those with an analytical bent, tend to think of FORMATs as kind of a luxury, something that is nice to have to make your output more reader-friendly, but by no means necessary to getting your work done in SAS. Over time I've come to appreciate the incredible flexibility and usefulness of FORMATs and the FORMAT procedure. FORMATs can do a lot more than simply beautify your output – though they are certainly handy for that too. Of course, many papers and at least one book have been written on the subject (a few handfuls of which are in the references section), and I can't even touch on all of them. Instead this paper is meant to serve as sort of my own Top Ten list – of some of the most valuable, and perhaps unexpected tasks you can achieve with SAS formats. They are not in order of importance or complexity. In nearly every example, I also refer you elsewhere to do more reading on that particular topic. As always, I also encourage you to experiment (and expand upon!) these techniques yourself.

This paper does assume some basic knowledge of SAS FORMATs and PROC FORMAT. If you need a primer or a review, in addition to the documentation directly from SAS, check out one or more of the following papers (Carpenter, 2004; Bilenas 2005; Bilenas 2008).

EXAMPLE #1: PICTURE FORMATS

It seems to me that PICTURE formats are underused. They are a way of providing a template or mask for the way values are displayed. Many of the built-in SAS formats, such as the many date formats or other numeric formats behave like PICTURE formats; that is, they don't specify precise text to display for a given value; rather they standardize the display – for example specifying leading zeros or a specific number of decimal places. Others have written whole papers on PICTURE formats (e.g. Croghan, 2004; Karp, 2006), which you should definitely look to for more syntax details and examples. Here I present a few that I use all the time.

Often in PROC TABULATE or REPORT, I would like percentages to display in parentheses (usually in a column next to the numerator). Another well-known "trick" I often use is to take the mean of a 0,1 indicator variable, which will correspond to the proportion of '1's' in the data, but I want that proportion to display as a number between 0 and 100 (i.e. like a percentage) rather than a number between 0 and 1. PICTURE formats are great for both of these types of tasks.

For this example, my data contain demographic and functional information on a ten percent sample of US Nursing Home residents. Here, I'm going to use variations on two variables.

- ***SEX*** has values of 1 for Male and 2 for Female, and has a very small number with missing data (assigned special missing value .M). For illustration, I also have a variable ***Male***, which is 1 if male, 0 if female and missing otherwise.
- ***Dementia3*** is a measure of level of cognitive impairment and has values of 0 for none to mild impairment, 1 for moderate impairment and 2 for severe impairment. There is a substantial amount of missing data, and

they are coded ***.M***. I also have a variable ***SevereImpairment***, which is 1 if the resident is severely impaired, 0 if none to mild or moderate, and missing otherwise.

I have used this code to create two VALUE FORMATs and two PICTURE FORMATs:

```
PROC FORMAT ;

VALUE dem3f 0 = 'None to Mild'
            1 = 'Moderate'
            2 = 'Severe'
           .M = 'Missing';

VALUE sex 1 = 'Male'
          2 = 'Female'
         .M = 'Missing' ;

PICTURE pct (ROUND)  low-high = '00009.99)'  (PREFIX = '(' );
PICTURE prop (ROUND) low-high = '00009.9'  (MULT = 1000) ;

RUN;
```

There is nothing special about the VALUE FORMATs; these will be used to customize the display of the levels of the CLASS variables. By contrast the two PICTURE FORMATs will be used to customize the display of the *statistics* requested in the TABLES statement of PROC TABULATE.

The ***pct*** PICTURE will be used to format the percentages, with two decimal places (because of the .99) and place them in parentheses. Note that the open parenthesis (which is before the numeric portion of the picture is specified in the PREFIX option, while the close parenthesis is part of the template itself. Because we specify the entire range of values ("low – high"), all values will be displayed with this template – we can refine this later – different templates can be used for different ranges, just as in a VALUE format. The leading zeros in the picture are used to specify the total width of the display – I want it wide enough that the column header (here, 'Percent') will not wrap. Finally, the ROUND option causes the value to be rounded (to the level of precision dictated by the template); without this the value would simply be truncated.

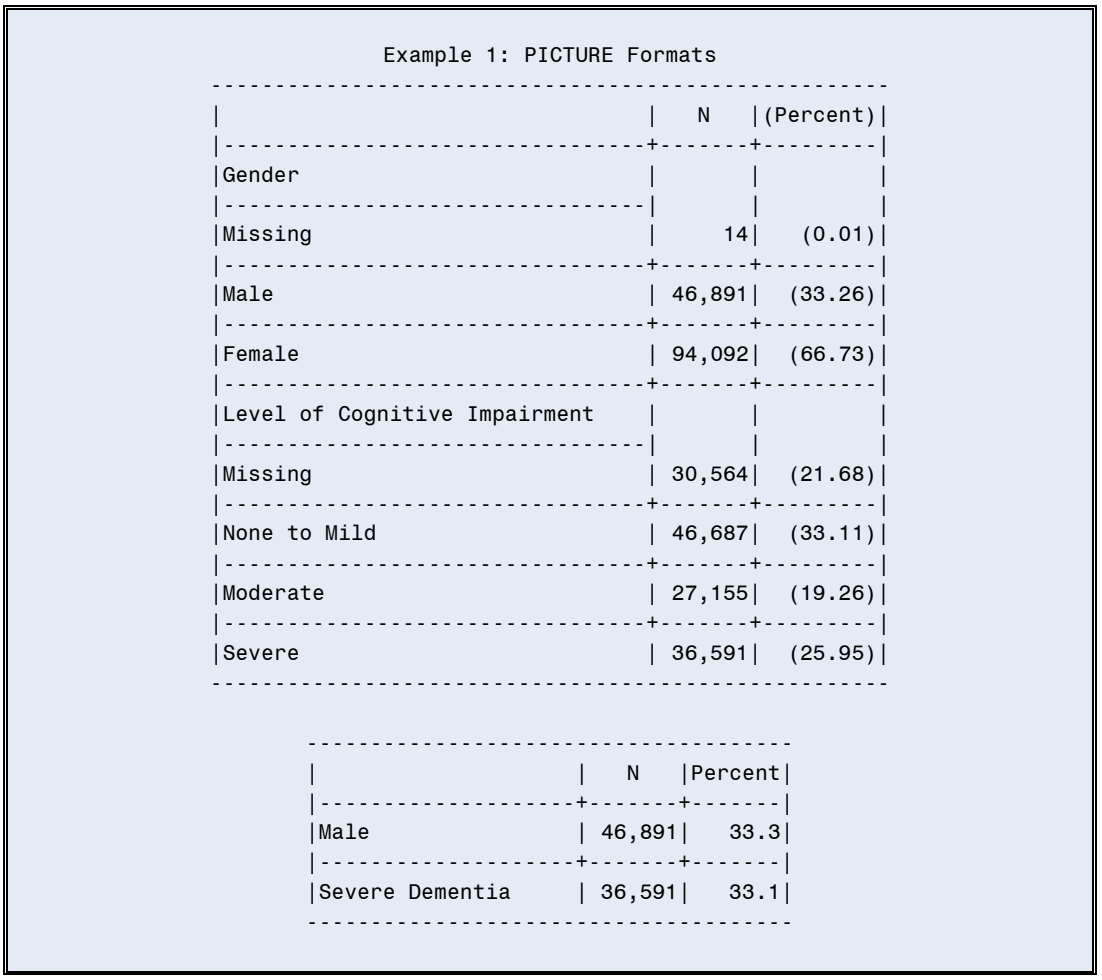
The ***prop*** PICTURE will be used to format the proportions; that is, the means of the binary variables, with a single decimal place. The MULT option specifies that the value of the statistic will be multiplied by 1000 before the PICTURE is applied. So, if the proportion is *0.325*, this will be converted to *325* and then the specification of one decimal place (.9) causes this to be displayed as *32.5*. If we wanted two decimal places, we would have .99 in the template but then would need to have MULT = 10000. (I always have to play with this a little to get it right! ☺). The ROUND option is also important here.

Here is the TABULATE code, in which these FORMATs are applied.

```
PROC TABULATE DATA = residents MISSING;
 CLASS sex dementia3 ;
 VAR Male SevereDementia ;
 TABLES (sex dementia3),(N*F=comma7. PCTN<sex dementia3>='(Percent)'*F=pct.)
              /RTS = 35;
 TABLES (Male SevereDementia),(SUM='N'*F=comma7.  MEAN='Percent'*F=prop.) ;
 FORMAT sex sex. dementia3 dem3f. ;
 TITLE1 'Example 1: PICTURE Formats';
 RUN;
```

Note that the VALUE formats (***sex*** and ***dem3f***) are used to modify the display of the levels of the CLASS variables (***sex*** and ***dementia3***) while the PICTURE formats (***pct*** and ***prop***) are used to modify the display of the PCTN and MEAN statistics, respectively.

The output (just to the listing destination) is shown in ***Output 1***. The reason that the percentage for severe dementia is different in the two tables is that in the first table the missing are included in the denominator (variable in CLASS statement with MISSING option in effect) while in the second they are excluded (variable in the VAR statement). For ***sex*** vs. ***male***, there is no difference simply because the number of missing is so small. I have just scratched the surface of what can be done with PICTURE formats. I hope this whets your appetite to play with them and search out more examples!



Example 1: PICTURE Formats

	N	(Percent)
Gender		
Missing	14	(0.01)
Male	46,891	(33.26)
Female	94,092	(66.73)
Level of Cognitive Impairment		
Missing	30,564	(21.68)
None to Mild	46,687	(33.11)
Moderate	27,155	(19.26)
Severe	36,591	(25.95)

	N	Percent
Male	46,891	33.3
Severe Dementia	36,591	33.1

Output 1. PICTURE formats have been used to customize the display of percentages and proportions in PROC tabulate.

EXAMPLE #2: USING FORMAT OPTIONS TO SPECIFY THE ORDER OF CLASS LEVELS

One thing I don't really like about the display in the first table of ***Output 1*** is that the missing category comes first for both the CLASS variables. This is because by default the order (at least with the TABULATE procedure) is based on the internal ordering (lowest to highest) of the underlying class level values, and missing is always less than anything else. I also want Female before Male. By making some small changes in the PROC FORMAT and TABULATE, we can get the order as desired.

```
PROC FORMAT ;
VALUE dem3f (NOTSORTED)
            0 = 'None to Mild'
            1 = 'Moderate'
            2 = 'Severe'
           .M = 'Missing' ;
```

```
VALUE sex (NOTSORTED)
          2 = 'Female'
          1 = 'Male'
         .M = 'Missing'   ;
RUN;

 PROC TABULATE DATA = residents MISSING ;
 CLASS sex dementia3 / ORDER=data PRELOADFMT ;
 VAR Male SevereDementia ;
 TABLES (sex dementia3),(N*F=comma7. PCTN<sex dementia3>='(Percent)'*F=pct.)
          /RTS = 35;
 FORMAT sex sex. dementia3 dem3f. ;
 TITLE1 'Example 2: Getting the order of class levels as desired';
 RUN;
```

By adding the NOTSORTED option to the VALUE statements AND placing the values in the desired order in the VALUE statement along with the ORDER=DATA and PRELOADFMT options on the CLASS statement in PROC TABULATE, I achieve the desired result (see ***Output 2***). Not all of these changes will always be required – depending on your data and the way the variables are coded; however, I have found that this combination is a safe bet to allow me to control the order of the CLASS level rows. I removed the second table as it wasn't needed for this example, but nothing else has changed in the code.

Example 2: Getting the order of class levels as desired

	N	(Percent)
Gender		
Female	94,092	(66.73)
Male	46,891	(33.26)
Missing	14	(0.01)
Level of Cognitive Impairment		
None to Mild	46,687	(33.11)
Moderate	27,155	(19.26)
Severe	36,591	(25.95)
Missing	30,564	(21.68)

Output 2. The combination of the NOTSORTED option on the VALUE statements in PROC FORMAT and PRELOADFMT and ORDER=data on the CLASS statement in PROC TABULATE put the levels of the CLASS variables in the desired order.

EXAMPLE #3: NESTING FORMATS

Suppose you have a bunch of standard codes that are used for many variables (e.g. different types of missing data) and then each variable also has its unique codes. This is a great use of nested formats! Suppose you want to superimpose a PICTURE format on top of a value format – another great use of nested formats! For this example, I have modified the data for the ***dementia3*** variable slightly so that there are several different categories of missing, which you can see below in the new ***missf***. FORMAT. I have also modified the ***dem3f*** format so that the ***missf***

format is nested within it – any values not explicitly listed in the ***dem3f*** VALUE statement (i.e. those falling into OTHER) will be formatted with the ***missf***. FORMAT. I have also created a new VALUE format called ***le1f.*** that will take any values less than one and label them with '(< 1.0); this format is then nested within the ***pct***. PICTURE format so that small percentages will show up as '(< 1.0)'. Specifying the default width of the format will prevent the percent column from getting a default width of 40. This is the best way I have found to assign specific strings as labels (as opposed to a template) to some values but apply the desired template to most of the range.

Here is the code:

```
PROC FORMAT ;
VALUE missf .N = 'N/A'
            .D = 'Don''t know'
           .R = 'Refused'
           .U = 'Unknown'
            .M = 'Missing'    ;
VALUE le1f (DEFAULT=9) 0 - <1 = '  ( <1.0)'  ;
VALUE dem3f (NOTSORTED)
            0 = 'None to Mild'
            1 = 'Moderate'
            2 = 'Severe'
        other = [missf.] ;
VALUE sex (NOTSORTED)
          2 = 'Female'
          1 = 'Male'
       other = [missf.]  ;
PICTURE pct (ROUND DEFAULT=9)  0 - <1 = [le1f.]
            1 - high = '00009.99)'  (PREFIX = '(' ) ;
RUN;

PROC TABULATE DATA = residents2 MISSING ;
CLASS sex dementia3a / ORDER=freq;
TABLES (sex dementia3a),(N*F=comma7. PCTN<sex dementia3a>='(Percent)'*F=pct.)
           / RTS = 35 ;
FORMAT sex sex. dementia3a dem3f. ;
TITLE1 'Example 3: Nesting Formats';
RUN;
```

Unfortunately, you cannot use PRELOADFMT with nested formats, so you might have to use other tricks to get the class levels in the desired order. In this case it works pretty well to use ORDER=freq. The result is shown in ***Output 3***.

Example 3: Nesting Formats

	N	(Percent)
Gender		
Female	94,092	(66.73)
Male	46,891	(33.26)
Missing	14	(<1.0)
Level of Cognitive Impairment		
None to Mild	46,687	(33.11)
Severe	36,591	(25.95)
Moderate	27,155	(19.26)
Missing	22,545	(15.99)
Don't know	4,576	(3.25)
Refused	3,021	(2.14)
N/A	422	(<1.0)

Output 3. By nesting formats, you can combine a standard set of codes (e.g. for different types of missing data) with variable-specific codes. You can also superimpose a PICTURE format on a VALUE format..

Before moving on to the next example, I will note that you can also nest SAS-supplied formats (e.g. date formats) within your user-defined formats. Lois Levin (Levin 2011) gives a great example, showing using several different date formats to provide different levels of precision on dates depending on the range.

EXAMPLE #4: PUTTING STYLE INFO INTO A FORMAT

Just in case you haven't seen this type of trick, I wanted to include this next one because I use it so frequently as a way to get my tables (usually from PROC REPORT or PROC TABULATE) to look the way I want in rtf output. This example and a few others make use of the ODS ESCAPECHAR. I'm not going to go into detail on that very broad topic – others have written great papers (e.g. Zender, 2007; Hadden, 2010) on it, but for these examples, assume that we have included the following statement in our code before SAS has to interpret the style information.

```
ODS ESCAPECHAR = '^' ;
```

In your PROC FORMAT VALUE statements you can then embed in-line style information using the ESCAPECHAR. These act as overrides of whatever other styles are in place (based on the style template you are using and whatever other style modifications are in your code), so you may have to experiment, but because they allow you to give different styles to the different values within a given format they give you a lot of flexibility. There are a bunch rtf control words and types of inline formatting that you can use (see, for example, Parsons 2007). Here I have modified the ***dem3f*** format so that the levels will be indented and the ***sex*** format so that these levels will be bold and indented. Note I have chosen not to change the style of the missing data so you can see what it will look like without adding these style elements. I am also inserting an rtf control word into the ***le1f*** format so that the "<1.0" will be italicized. The '^' ESCAPECHAR signals that some in-line formatting is coming and the '\' signals the use of an rtf control word.

The combination of these two is needed before each change, so the code can end up looking pretty strange. In general the '0' after a formatting command turns it off – so, for example, the 'i0' turns off italics (which is the default of the style template I was using). I particularly like being able to add indentation in my TABULATE output – so I use the '^\~' a lot. The RTF "sandwich" and the PROC TABULATE code are also shown.

```
PROC FORMAT ;

VALUE dem3f (NOTSORTED)
            0 = '^\i0^\~ None to Mild'
            1 = '^\i0^\~ Moderate'
           2 = '^\i0^\~ Severe'
       .M = 'Unknown'
           ;
VALUE sex (NOTSORTED)
          2 = '^\~^\b Female'
      1 = '^\~^\b Male'
      .M = 'Unknown'
      ;
VALUE le1f (DEFAULT=9) 0 - <1 = '^\i(<1.0)'  ;

PICTURE pct (ROUND DEFAULT=9)  0 - <1 = [le1f.]
                     1 - high = '00009.99)'  (PREFIX = '(' ) ;
RUN;

ODS RTF FILE='Example4.rtf'  PATH=odsout STYLE=journal bodytitle;
ODS ESCAPECHAR = '^';

PROC TABULATE DATA = residents MISSING ;
 CLASS sex dementia3  /ORDER=formatted;
 TABLES (sex dementia3),(N*F=comma7. PCTN<sex dementia3>='(Percent)'*F=pct.)
          /RTS =35 ;
 FORMAT sex sex. dementia3 dem3f. ;
 TITLE1 'Example 4: Putting Style information into Formats';
 RUN;

ODS RTF CLOSE ;
```

The result (from rtf) is shown in ***Output 4***. Of course, there is plenty more that you could do with styles in the PROC TABULATE code to enhance the way the table looks, but I wanted to focus on the action of the FORMATs here.

Example 4: Putting Style information into Formats

	N	*(Percent)*
Gender		
Unknown	14	*(<1.0)*
Female	94,092	(66.73)
Male	46,891	(33.26)

Example 4: Putting Style information into Formats

	N	*(Percent)*
Level of Cognitive Impairment		
Unknown	30,564	(21.68)
Moderate	27,155	(19.26)
None to Mild	46,687	(33.11)
Severe	36,591	(25.95)

Output 4. Using ODS ESCAPECHAR and rtf control words within the VALUE and PICTURE formats allows you to add styles, such as indentation and boldface, to specific ranges.

EXAMPLE #5: PUTTING UNICODE CHARACTERS INTO A FORMAT

This next example is just kind of a fun one, with the objective to introduce you to Unicode characters. As of version 9.2, you can use Unicode characters to put symbols into your output – and FORMATS. Louise Hadden (2010) explains how to use the Windows character map to find the correct Unicode values. The syntax is ^{Unicode xxxx} where ^ is the specified ODS ESCAPECHAR and xxxx is the 4-digit Unicode value. I show two examples here. In the first, I add the male and female symbols to the gender rows. Here is the code:

```
PROC FORMAT ;
VALUE sex 1 = '^{Unicode 2642} Male'
          2 = '^{Unicode 2640} Female'
        .M = 'Unknown'
      ;
ODS RTF FILE='Example5.rtf'  PATH=odsout STYLE=journal bodytitle;

ODS ESCAPECHAR = '^';

PROC TABULATE DATA = residents  ;
 CLASS sex   /ORDER=formatted;
 TABLES (sex ),(N*F=comma7. PCTN<sex>='(Percent)'*F=pct.) /RTS =35 ;
 FORMAT sex sex.  ;
 TITLE1 'Example 5: Using Unicode Symbols in Formats';
 RUN;

ODS RTF CLOSE ;
```

For simplicity I have left the missing out and am just showing gender in ***Output 5a***.

Example 5: Using Unicode Symbols in Formats

	N	*(Percent)*
Gender		
♀ *Female*	94,092	(66.73)
♂ *Male*	46,891	(33.26)

Output 5a. Unicode symbols are placed in the levels of the VALUE format in order to use special characters (here the male and female symbols) in the table.

There are literally hundreds of symbols available – including, Greek letters, mathematical symbols and many other special characters – so check out the character map! The way I most commonly use this technique is for the reports I do on nursing home ratings. Here we want to show the ratings (which have underlying values of 1-5) as stars. In our rtf reports we do this with Unicode. Here I just show the format – see that the ESCAPECHAR is needed before each instance of the Unicode value -- and then a snippet of a report (***Output 5b***).

```
PROC FORMAT;

VALUE starfmt
 1='^{Unicode 2605}'
 2='^{Unicode 2605}^{Unicode 2605}'
 3='^{Unicode 2605}^{Unicode 2605}^{Unicode 2605}'
 4='^{Unicode 2605}^{Unicode 2605}^{Unicode 2605}^{Unicode 2605}'
 5='^{Unicode 2605}^{Unicode 2605}^{Unicode 2605}^{Unicode 2605}^{Unicode 2605}';
RUN;

ODS ESCAPECHAR = '^';
ODS RTF FILE='myfile.rtf' PATH=odsout STYLE=journal;

PROC TABULATE DATA = …. ;
/* < CODE SNIPPED OUT HERE > */
FORMAT rating starfmt. ;
RUN;
```

Table 1. Distribution of Nursing Home Ratings, All Nursing Homes, August 2014

5-star Measure	★		★★		★★★		★★★★		★★★★★	
	N	(%)	N	(%)	N	(%)	N	(%)	N	(%)
Overall Rating	1,435	(9.3)	3,179	(20.5)	2,670	(17.2)	3,982	(25.7)	4,238	(27.3)
Health Inspections	3,080	(19.9)	3,556	(22.9)	3,557	(22.9)	3,624	(23.4)	1,687	(10.9)
MDS Quality Measures	363	(2.3)	961	(6.2)	2,211	(14.3)	5,422	(35.1)	6,494	(42.0)
Staffing	1,677	(11.0)	2,176	(14.3)	2,949	(19.4)	6,643	(43.8)	1,739	(11.5)
RN Staffing	1,448	(9.5)	2,386	(15.7)	3,959	(26.1)	4,056	(26.7)	3,335	(22.0)

Output 5b. Putting Unicode characters into a VALUE format allows the starts to shine in my rtf output.

EXAMPLE #6: MULTILABEL FORMATS

Multilabel FORMATs are extremely useful. They allow you to display overlapping or repeated ranges in your tables and can be used in PROC TABULATE, PROC REPORT, and PROC MEANS/SUMMARY. This capability has been in SAS since version 8, but I think it deserves highlighting. It is invaluable for producing subtotals in tables! Here is one simple example with the age of nursing home residents. In the VALUE statement of PROC FORMAT, the MULTILABEL option is required to specify a format that may have repeated or overlapping ranges. In the CLASS statement, you also need to specify MLF to indicate a multilabel format. The combination of NOTSORTED (in PROC FORMAT) and ORDER=DATA and PRELOADFMT (in the CLASS statement) will ensure that the rows come out in the desired order in the table. In the code below the format ***agemlfa***. is a basic multilabel format, while ***agemlfb***. takes it up a notch with some inline formatting to make it more clear in the output that there are subtotals and to get the ≤ and ≥ symbols where desired.

```
PROC FORMAT;
VALUE agemlfA (NOTSORTED MULTILABEL)
 0-21 = '0-21'
 22-30 = '22-30'
 31-64 = '31-64'
 0-64 = '<65'
 65-74 = '65-74'
 75-84 = '75-84'
 85-94 = '85-94'
 95-high = '95+'
 65-high = '65+'
 85-high = '85+'  ;

VALUE agemlfB (NOTSORTED MULTILABEL)
 0-21 = '^\~^\ul<^\ul0 21'
 22-30 = '^\~22-30'
 31-64 = '^\~31-64'
 0-64 = '^\b < 65'
 65-74 = '^\~65-74'
 75-84 = '^\~75-84'
 85-94 = '^\~85-94'
 95-high = '^\~^\ul>^\ul0 95'
 65-high = '^\b^\ul>^\ul0 65'
 85-high = '^\b^\ul>^\ul0 85'  ;
RUN;

ODS RTF FILE='Example6.rtf'  PATH=odsout STYLE=journal bodytitle;

ODS ESCAPECHAR = '^';
PROC TABULATE DATA = residents  ;
 CLASS age   /MLF PRELOADFMT ORDER=DATA;
 TABLES (age=' ' ),(N*F=comma7. PCTN<age>='(Percent)'*F=pct.)
        /RTS =35 BOX='Resident Age';
 FORMAT age agemlfa.   ;
 TITLE1 'Example 6: Multilabel Formats';
 RUN;

 PROC TABULATE DATA = residents  ;
 CLASS age   /MLF PRELOADFMT ORDER=DATA;
 TABLES (age=' ' ),(N*F=comma7. PCTN<age>='(Percent)'*F=pct.)
      /RTS =35 BOX='Resident Age';
 FORMAT age agemlfb.   ;
 TITLE1 'Example 6: Multilabel Formats - Enhanced';
 RUN;

ODS RTF CLOSE ;
```

Results using both of these formats are shown in ***Output 6***.

Example 6: Multilabel Formats

Resident Age	*N*	*(Percent)*
0-21	285	*(<1.0)*
22-30	467	*(<1.0)*
31-64	20,488	(14.53)
<65	21,240	(15.07)
65-74	21,256	(15.08)
75-84	38,182	(27.09)
85-94	49,377	(35.03)
95+	10,904	(7.74)
65+	119,719	(84.93)
85+	60,281	(42.76)

Example 6: Multilabel Formats - Enhanced

Resident Age	*N*	*(Percent)*
≤21	285	*(<1.0)*
22-30	467	*(<1.0)*
31-64	20,488	(14.53)
< 65	21,240	(15.07)
65-74	21,256	(15.08)
75-84	38,182	(27.09)
85-94	49,377	(35.03)
≥95	10,904	(7.74)
≥65	119,719	(84.93)
≥85	60,281	(42.76)

Output 6. Multilabel formats allow you to include subtotals in the table. It is also handy to combine them with in-line formatting techniques.

EXAMPLE #7: CONVERTING A DATA SET TO A FORMAT (CNTLIN= OPTION)

No paper on FORMATs would be complete without a discussion of the CNTLIN=option, which allows you to convert a data set into a format. This can be very handy for doing table look-ups and other MERGE-like tasks. The way that I have used this most commonly is when I have data that has a variable with a large number of possible codes, and I

want to report on these codes but assign some text meaning to them. Common examples in healthcare would be diagnosis codes or drug codes; in business, you might have part numbers. Sticking with my nursing home world, I'm going to use the examples of nursing home health inspection results, which include four-digit numeric codes (called a "tag") corresponding to one of several hundred regulations that the nursing home could be found to be violating. The numbers themselves have very little meaning to most people, so for a report to be useful, they would need to be labeled with some descriptive text. This is, of course, a perfect task for a FORMAT; however, who wants to type all those labels into her program?! If you have (or can generate) a data set that matches the code to the text description then you can use PROC FORMAT with the CNTLIN= option to turn that data into a format.

Figure 1 shows a small portion of the descriptive information on these regulations. I originally received this in an Excel file, which I read into a SAS data set called TAG_DESCRIP.

TAG	DESCRIPTION
0151	Honor residents' rights as residents of the nursing home, free of coercion and reprisal, and as citizens or residents of the United States.
0152	Give the resident's representative the ability to exercise the resident's rights.
0153	Let each resident or the resident's legal representative access or purchase copies of all the resident's records.
0154	Ensure that residents are fully informed and understand their health status, care and treatments.
0172	Allow residents to have visitors.
0221	Keep each resident free from physical restraints, unless needed for medical treatment.
0222	Keep each resident free from drugs that restrain them, unless needed for medical treatment.
0223	Protect each resident from all abuse, physical punishment, and involuntary separation from others.
0224	Protect each resident from mistreatment, neglect and misappropriation of personal property.

Figure 1. Small subset of potential nursing home health inspection citations, to be converted into a FORMAT using the CNTLIN= option.

The code below builds the FORMAT from the data set and then uses it in a simple TABULATE. The CNTLIN option requires that the data set from which you are building a format has a few specific features, and the DATA step shown below makes sure the data set is ready for use by PROC FORMAT. Specifically, the CNTLIN data set must have a variable called START that indicates the start of the range. If no END variable is specified then each range will have a single value (as desired here). There must also be a variable called LABEL that contains the text to which each range will be mapped. The variables containing this information are already on the ***tag_descrip*** data set; so I simply rename then. The variable FMTNAME must supply the name of the FORMAT to be created, and TYPE here indicates that it is a character format we are creating. Alternatively, if you put a dollar sign before the FMTNAME (i.e. RETAIN FMTNAME '$tag_desc'), PROC FORMAT will know you are creating a character format. Using RETAIN to assign these values to all the observations on the file is more efficient than assignment statements, though the latter would also work. To be tidy, I KEEP only those variables, but it is not a problem for the FORMAT procedure if there are other variables on the file.

The PROC FORMAT code is exceedingly simple – all you do is tell it the name of the dataset containing the format info. Note that the information for multiple formats could be in the same CNTLIN data set – the variable FMTNAME would tell SAS to which format each row of the data belonged.

```
DATA cntl_tagdesc ;
 SET tag_descrip (RENAME = (tag=START tag_description=LABEL))  ;

RETAIN FMTNAME 'tag_desc'
       TYPE 'C';

KEEP FMTNAME TYPE START LABEL;
RUN;

PROC FORMAT CNTLIN=cntl_tagdesc ;
RUN;
```

```
PROC TABULATE DATA = defs.alldefs20140801 ORDER=FREQUENCY;
CLASS tag ;
 TABLES (tag =' '),(N*F=comma7. PCTN<tag>='(Percent)'*F=pct.)
      /RTS =45 BOX='Deficiency';
FORMAT TAG $tag_desc. ;
TITLE 'Example 7 - Turning a DATA set into a FORMAT';
RUN;
```

I can then use this FORMAT as I would any other format – Here I use it in a PROC TABULATE; a very small portion of the output is shown in ***Output 7***.

Example 7 - Turning a DATA set into a FORMAT

Deficiency	N	(Percent)
Have a program that investigates, controls and keeps infection from spreading.	5,773	(6.49)
Store, cook, and serve food in a safe and clean way.	5,485	(6.16)
Ensure that a nursing home area is free from accident hazards and provide adequate supervision to prevent avoidable accidents.	4,581	(5.15)
Provide necessary care and services to maintain or improve the highest well being of each resident .	3,828	(4.30)
Ensure that each resident's 1) entire drug/medication regimen is free from unnecessary drugs; and 2) is managed and monitored to achieve highest level of well-being.	3,586	(4.03)
Maintain drug records and properly mark/label drugs and other similar products according to accepted professional standards.	3,152	(3.54)

Output 7. A small portion of the table showing the frequency of nursing home health inspection citations, formatting the codes corresponding to the citations using a format generated with the CNTLIN=option.

EXAMPLE #8: USING A FORMAT FOR DATA CLEANING

This example is an extension of the previous one. The large data set that contains the health inspection results (a portion of which is tabulated in ***Output 7***) may have some invalid tag numbers, perhaps because of data entry errors. I want to identify these – either for exclusion or to kick back to a data manager for correction or, perhaps, to learn that there are new tags for which I need to get descriptions. Using the FORMAT I created in Example 7, and then the TABULATE, these would just come out as the (invalid) numeric codes in the table, but let's say I want to output them to another data set or simply generate a list of them. One way to do this is first to modify the format slightly so that any value NOT in my look-up list gets assigned a specific label. Then, I can identify the observations in my data that get mapped to that label.

In the code below I simply add one more observation to the CNTLIN data set, which basically corresponds to the "OTHER=" portion of a format created with a VALUE statement. Effectively this will map any value that is not in the

data set to the label "***INVALID***". The PROC FORMAT is identical to the previous example. Then, in the subsequent DATA step, I apply the ***$tag_desc.*** format to all the TAGs in the health inspection data and output only those that get flagged as invalid. This is an extremely useful technique for subsetting data and can be much more efficient than a MERGE (see an example in Levin, 2011). Also, for a complete description of an application of PROC FORMAT for a complex data validation and cleaning task, see Williams 2007

```
DATA cntl_tagdesc2 ;
 SET tag_descrip (RENAME = (tag = START tag_description=LABEL))  END=lastobs;

RETAIN FMTNAME 'tag_desc'
       TYPE 'C';

OUTPUT ;

* if a tag doesn't get mapped to a description, we want to flag it;
IF lastobs THEN DO;
   HLO='O';
   LABEL='***INVALID***';
   OUTPUT ;
END;

KEEP FMTNAME TYPE START LABEL HLO;
RUN;

PROC FORMAT CNTLIN = cntl_tagdesc2 ;
RUN;

DATA invalid_tags ;
  SET defs.alldefs20140801;

  WHERE PUT(tag,$tag_desc.) = '***INVALID***';
RUN;
```

EXAMPLE #9: CREATE A PERMANENT FORMAT LIBRARY AND GIVE IT A USEFUL NAME

These last two examples deal with managing format libraries. It is a pet peeve of mine that so many format libraries end up with the extremely generic name FORMATS.SAS7BCAT. This is, of course, the default name given to a permanent FORMAT catalog, if you use the following type of code:

```
* Please don't do this !! ;
LIBNAME library 'c:\mylibrary\';

PROC FORMAT LIBRARY=library;

VALUE sex 1 = 'Male'
          2 = 'Female'
        .M = 'Missing'  ;
RUN;
```

If you execute the above code, the SAS log will tell you that "*NOTE: FORMAT sex has been written to LIBRARY.FORMATS*", and you will see a file in that directory called FORMATS.SAS7BCAT. First, you do NOT have to use the libref "library" for a format library. You can use any valid libref. Secondly, and much more importantly, you can use a two-level name in the PROC FORMAT statement so that the format catalog created can have a more informative name! You can have more than one format library in the same directory, distinguished by their names

(what a concept!) and they can even contain different formats with the same names (though this is probably not a good idea). You tell SAS which libraries to look in for formats with the FMTSEARCH option. Here is how you do this:

```
LIBNAME fmthome '.';

PROC FORMAT LIBRARY = fmthome.hlthinspec2014 CNTLIN = cntl_tagdesc;
RUN;

PROC FORMAT LIBRARY = fmthome.hlthinspec2014 ;
VALUE sex 1 = 'Male'
          2 = 'Female'
      .M = 'Missing'
      ;
RUN;

PROC FORMAT LIBRARY=fmthome.my_rtf_fmts ;

VALUE agemlfB (NOTSORTED MULTILABEL)
 0-21 = '^\~^\ul<^\ul0 21'
 22-30 = '^\~22-30'
 31-64 = '^\~31-64'
 0-64 = '^\b < 65'
 65-74 = '^\~65-74'
 75-84 = '^\~75-84'
 85-94 = '^\~85-94'
 95-high = '^\~^\ul>^\ul0 95'
 65-high = '^\b^\ul>^\ul0 65'
 85-high = '^\b^\ul>^\ul0 85'
     ;
VALUE dem3f
            0 = '^\i0^\~ None to Mild'
            1 = '^\i0^\~ Moderate'
           2 = '^\i0^\~ Severe'
       .M = 'Unknown'
           ;
VALUE sex 1 = '^{Unicode 2642} Male'
          2 = '^{Unicode 2640} Female'
      .M = 'Unknown'
      ;

value starfmt
 1='^{Unicode 2605}'
 2='^{Unicode 2605}^{Unicode 2605}'
 3='^{Unicode 2605}^{Unicode 2605}^{Unicode 2605}'
 4='^{Unicode 2605}^{Unicode 2605}^{Unicode 2605}^{Unicode 2605}'
 5='^{Unicode 2605}^{Unicode 2605}^{Unicode 2605}^{Unicode 2605}^{Unicode 2605}'
;

RUN;
```

The LIBNAME statement assigns a libref for where I'm going to put the formats. Of course, the libref ***fmthome*** is arbitrary. The subsequent PROC FORMAT puts the ***tag_descrip*** format (created in the previous example) into a FORMAT catalog called ***hlthinspec2014.sas7bcat*** into the directory specified by the LIBNAME statement. The next PROC FORMAT step simply shows that you can add more formats to the library. If there was already a format called ***sex***. in that library, it would be overwritten. The next PROC FORMAT step puts several formats into another format catalog called ***my_rft_fmts.sas7bcat***, which happens to live in the same directory as the hlthinspec2014.sas7bcat – with a different top-level name (with an associated LIBNAME statement), it could go somewhere else.

Use the FMTSEARCH= option to specify what directories or specific format libraries SAS should search for FORMATS that you use (and in what order). Consider the two following examples:

```
OPTIONS FMTSEARCH = (work fmthome);
OPTIONS FMTSEARCH = (work fmthome.hlthinspec2014);
```

The first statement will tell SAS to first look in the temporary work library and then to look in any/all format catalogs that are in the directory associated with the libref FMTHOME. In contrast, the second statement tells SAS to first look in the temporary work library and then to look in only the hlthinspec2014 format catalog. I think it is good practice to put WORK first in the search string because you typically would want to give newly generated formats (from the current SAS session/job) precedence over others with the same name (should they exist elsewhere). In general giving your format libraries (and the formats in them!) useful names AND specifying the search path for formats will help you manage your formats and make sure you have access to the ones you want!

EXAMPLE #10: GIVE YOUR FORMATS HELPFUL DESCRIPTIONS

While it has been recommended for a long time (see e.g. Shoemaker 2000), I think that the feature of PROC CATALOG that allows you to provide a description for your formats is underutilized and appreciated. It is not hard to do!

```
PROC CATALOG CATALOG = fmthome.hlthinspec2014 ;
 MODIFY sex.format (DESCRIPTION = 'Format for Resident Gender' );
 MODIFY tag_desc.formatc (DESCRIPTION = 'Map Tags to Text Descriptions') ;
RUN;

PROC CATALOG CATALOG=fmthome.hlthinspec2014 ;
CONTENTS;
TITLE 'Example 10.  Giving Descriptions to FORMATs';
RUN;

PROC FMTLIB LIBRARY= fmthome.hlthinspec2014 ;
TITLE 'Example 10.  Formats in Hlthinspec2014 library';
RUN;
```

The two invocations of PROC CATALOG could be combined into a single step, but I wanted to separately show the syntax for giving the FORMATS descriptions and then displaying those descriptions. Note that in the MODIFY statements, you need to specify whether it is a numeric or character format, designated as FORMAT or FORMATC respectively. In fact, it is OK to have a numeric and a character format with the same name (distinguished, of course, by the presence of the $ when the character format is referenced. The second PROC CATALOG with the CONTENTS statement generates the output shown in ***Output 10a.*** Finally, a listing of the FORMATS themselves can be generated with the FMTLIB procedure, and a portion of this output is also shown in ***Output 10b***.

```
                        Example 10.  Giving Descriptions to FORMATs

                            Contents of Catalog FMTHOME.HLTHINSPEC2014

#   Name       Type          Create Date      Modified Date   Description
-------------------------------------------------------------------------------------------
1 SEX       FORMAT   08/08/2014 22:17:54  08/08/2014 23:08:50  Format for Resident Gender
2 TAG_DESC  FORMATC  08/08/2014 22:17:54  08/08/2014 23:08:50  Map Tags to Text Description
```

Output 10a. Use PROC CATALOG to give descriptions to FORMATs and to display a list of formats.

Example 10. Formats in Hlthinspec2014 library

```
-------------------------------------------------------------------------
|       FORMAT NAME: SEX      LENGTH:    7   NUMBER OF VALUES:    3      |
|   MIN LENGTH:   1  MAX LENGTH:  40  DEFAULT LENGTH:   7  FUZZ: STD     |
|-----------------------------------------------------------------------|
|START           |END             |LABEL  (VER. V7|V8   08AUG2014:22:17:54)|
|----------------+----------------+---------------------------------------|
|              .M|              .M|Missing                                |
|               1|               1|Male                                   |
|               2|               2|Female                                 |
-------------------------------------------------------------------------

-------------------------------------------------------------------------
|                    FORMAT NAME: $TAG_DESC LENGTH: 262                   |
|   MIN LENGTH:   1  MAX LENGTH: 262  DEFAULT LENGTH: 262  FUZZ:       0  |
|-----------------------------------------------------------------------|
|START           |END             |LABEL  (VER. 9.4     08AUG2014:22:17:54)|
|----------------+----------------+---------------------------------------|
|0150            |0150            |Meet the legal definition of a skilled n|
|0151            |0151            |Honor a residents' rights as a residents|
|0152            |0152            |Give the resident's representative the a|
|0153            |0153            |Let each resident or the resident's lega|
|0154            |0154            |Ensure that residents are fully informed|
|0155            |0155            |Let residents refuse treatment, refuse t|
 ...                      ...            ...
```

Output 10b. Use PROC FMTLIB to provide a detailed listing of the FORMATS in a particular CATALOG.

It is a "feature" of PROC FMTLIB that the display of the LABELs will be truncated in the output. Rest assured that the complete text is included in the format – and test it by using PROC FORMAT with the CNTLOUT= options to convert the format information back into a data set!

CONCLUSIONS

As I wrote this paper, I came to realize just how many papers have already been written about PROC FORMAT and user-defined formats! After all, this procedure has been around just about as long as SAS. So, I can't claim that I have great new insights about this well-mined topic. However, I hope it is of benefit to bring a bunch of these features together in one place, providing some specific ways in which I use formats on a daily basis. It never hurts to see more examples! To that end, I have also provided quite a few references to others' work, where you can find additional examples and further details on some of these methods. I do encourage you to get to know more about how FORMATs can strengthen your SAS programming proficiency.

REFERENCES AND RECOMMENDED READING

Bilenas Jonas. 2008. "I Can Do that with PROC FORMAT" *Proceedings of SAS Global Forum 2008..* Available at http://www2.sas.com/proceedings/forum2008/174-2008.pdf

Bilenas Jonas. 2005 "The Power of PROC FORMAT; Updated for SAS 9" *Proceedings of NESUG 2005.* Available at http://www.lexjansen.com/nesug/nesug05/pm/pm6.pdf.

Carpenter, Art. 2004 "Building and Using User-Defined Formats" *Proceedings of SUGI 29*. Available at http://www2.sas.com/proceedings/sugi29/236-29.pdf.

Croghan, Cary 2004. “PICTURE Perfect: In depth look at the PICTURE format” *Proceedings of SESUG 2004*. Available at http://analytics.ncsu.edu/sesug/2004/TU03-Croghan.pdf.

Zender, Cynthia 2007. “Funny ^Stuff~ in My Code: Using ODS ESCAPECHAR.”. *Proceedings of SAS Global Forum 2007*. Available at http://www2.sas.com/proceedings/forum2007/099-2007.pdf.

Hadden, Louise, 2010 “The Great Escape(char)” *Proceedings of SAS Global Forum 2010*. Available at http://support.sas.com/resources/papers/proceedings10/215-2010.pdf

Karp, Andrew 2006 “Getting in to the Picture (Format)”. *Proceedings of SUGI 31*. Available at http://www2.sas.com/proceedings/sugi31/243-31.pdf

Levin, Lois 2011. PROC FORMAT – Not Just Another Pretty Face. *Proceedings of NESUG 2011*. Available at http://www.lexjansen.com/nesug/nesug11/pf/pf04.pdf

Parsons Lori 2007. “Enhancing RTF Output with RTF Control Words and In-Line Formatting “ *Proceedings of SAS Global Forum 2007*. Available at http://www2.sas.com/proceedings/forum2007/151-2007.pdf

Williams, Christianna. 2007. “Using PROC FORMAT for DATA Validation and Clean-up “ Proceedings of SAS Global Forum 2007. Available at http://www2.sas.com/proceedings/forum2007/240-2007.pdf

CONTACT INFORMATION

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

Name: Christianna Williams
E-mail: Christianna.S.Williams@gmail.com