

Programming Solutions with SAS® Formats and Informats

Imelda C. Go, Richland County School District One, Columbia, SC

ABSTRACT

Formats and informats are programming tools that can simplify programming tasks. This paper demonstrates how these tools can be used to more easily accomplish certain tasks. The solutions these tools provide are simple; result in an economy of code; increase the readability of programs; and take advantage of built-in SAS features that facilitate data processing, reporting, and general programming. Topics covered include recoding and look-up tables.

INTRODUCTION

The purpose of this paper is to provide an overview of how SAS formats and informats can be used to simplify certain programming tasks. Examples in this paper provide programming statements that accomplish a task using formats/informats. Many of these examples provide alternative programming statements that accomplish the same task without using formats/informats.

STANDARD FORMATS AND INFORMATS

The fundamental difference between a format and an informat is indicated by the terminology. Informats help process input data (as the prefix *in-* implies). Formats on the other hand, help display data in SAS output. There are several built-in or *standard* SAS formats and informats. These allow data to be read and written in a prescribed manner.

In the following example, grades range from 0 to 4 (0.0, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0). However, the raw data provided below range from 0 to 40 and have an implied decimal place. Consider the following methods of processing the data:

<pre>**METHOD 1; data grades; input grade 2.1; cards; 00 10 15 20 25 30 35 40 ;</pre>	<pre>**METHOD 2; data grades; input grade; grade=grade/10; cards; 00 10 15 20 25 30 35 40 ;</pre>
---	---

Both methods produce the same data set and produce the following PROC PRINT output.

OBS	GRADE
1	0.0
2	1.0
3	1.5
4	2.0
5	2.5
6	3.0
7	3.5
8	4.0

The first method is possible because there is a known rule that can be used to read the data (i.e., all data have an implied decimal place). The standard SAS informat of 2.1 was used to read the data. The 2.1 informat is used with a numeric value consisting of 2 digits (width of 2 spaces) with 1 decimal place. This method requires knowing the exact form of the data in order to identify the *pattern* or informat that can be used to read the data.

On the other hand, the second method uses a computational statement to achieve the same result without using an informat. There are clear advantages to using the first method: the data are processed correctly from the beginning, the data do not require statements that manipulate data values later, and it is easier to check the programming code.

It is worth reviewing the standard SAS formats and informats. There are standard formats and informats related to real numbers, integers, percentages, currency, dates, time, and character values. In spite of this variety, there may be a need to create non-standard or user-defined formats or informats. This is possible by using PROC FORMAT.

USER-DEFINED FORMATS AND INFORMATS

Assume that a data set contains grade values that range from 0 to 40 instead of from 0 to 4.0. It is not important that the values in the data set are in the range of 0 to 40, but it is important that these values are printed out in the range of 0 to 4.0. A SAS picture format makes this possible.

A SAS picture format can only be applied to numeric variables. It specifies a pattern or template for the appearance of the numbers in SAS output without altering the actual values of the variable as they are stored in the data set. There are picture format options that control the appearance of leading zeroes, fill characters, and prefixes.

```
data grades;
input grade;
cards;
00
10
15
20
25
30
35
40
;

proc format;
  picture grade low-high='9.9' (multiplier=1);

proc print;
  var grade;
  format grade grade.;
```

The picture format `grade.` makes it possible to print the grades in the range of 0 to 4.0 even if the internal values range from 0 to 40.

GENERAL PROC FORMAT SYNTAX

The general syntax for VALUE, INVALUE, and PICTURE statements is shown below. (Other options and statements used with PROC FORMAT are not included.)

```
PROC FORMAT <option-list>;
  VALUE name<(format-option-list)>
    range-1='formatted-value-1'
    <...range-n>='formatted-value-n'>;
  INVALUE name<(informat-option-list)>
    <'>range-1<'>=informatte-value-1
    <...<'>range-n<'>=informatte-value-n>;
  PICTURE name<(format-option-list)>
    range-1=picture-1' <(picture-option-list)>;
    <...range-n=picture-n' <(picture-option-list)>>;
```

Informats are specified using INVALUE statements. Formats are specified using VALUE or PICTURE statements. Several VALUE, INVALUE, or PICTURE statements can be specified under one PROC FORMAT statement.

SOME SYNTAX NOTES FOR INFORMATS

Character informat names begin with a dollar sign (\$) while numeric informat names do not. A *character informat is used when the resulting variable needs to be a character variable. A numeric informat is used when the resulting variable needs to be a numeric variable.*

```
proc format;
invalue $rating      'A'=excellent      'B+=very good
                    'B'=good           'C+=above average
                    'C'=average         'D+=below average
                    'D'=poor            'F'=failure;
invalue $words       'A'=excellent'    'B+=very good'
                    'B'=good'          'C+=above average'
                    'C'=average'        'D+=below average'
                    'D'=poor'           'F'=failure';
invalue $one         'A'=4              'B+=3.5
                    'B'=3               'C+=2.5
                    'C'=2               'D+=1.5
                    'D'=1               'F'=0;
invalue $two         'A'='4'            'B+'='3.5'
                    'B'='3'            'C+'='2.5'
                    'C'='2'            'D+'='1.5'
                    'D'='1'            'F'='0';
invalue $letter      4='A'              3.5='B+'
                    3='B'              2.5='C+'
                    2='C'              1.5='D+'
                    1='D'              0='F';
invalue three        'A'=4              'B+=3.5
                    'B'=3              'C+=2.5
                    'C'=2              'D+=1.5
                    'D'=1              'F'=0;
invalue group        2-high=1           low-1.5=0;
invalue grp          2-4=1              0-1.5=0;
```

For character informat \$rating., the informatte values were not enclosed in quotes, but SAS considers these values as enclosed in quotes. Hence, informat \$rating. and \$words. are equivalent and so are \$one. and \$two.. Informats three., group., and grp. are numeric informats. The informat group. uses the keywords LOW and HIGH. The keyword LOW refers to the *lowest* value of a variable. (The range low-1.5 refers to values that range from the lowest value of the variable till 1.5.) The keyword HIGH refers to the *highest* value of a variable. (The range 2-high refers to values that range from 2 till the highest value of the variable.)

Consider the following examples:

- ♦ The value of A will be read as excellent and grade is a character variable.

```
data convert;
input grade $words.;
cards;
A
;
```
- ♦ The value of 4 will be read as A and grade is a character variable.

```
data convert;
input grade $letter.;
cards;
4
;
```
- ♦ The value of A will be read as 4 and grade is a numeric variable.

```
data convert;
input grade three.;
cards;
A
;
```
- ♦ The value of 3.5 will be read as 1 and grade is a numeric variable.

```
data convert;
input grade group.;
cards;
3.5
;
```

What happens when an attempt is made to designate a character value as an informatte value for a numeric format?

```
proc format;
invalue group      2-high='1'          low-1.5=0;
```

This generates the following error message:

```
The quoted string '1' is not acceptable to a
numeric format or informat.
```

What happens when an attempt is made to apply an informat to a value that was not specified in any informatte range? The unspecified value will not be converted, but will remain the same as long as there is no conflict between the value and the resulting variable type.

In the following example, the value of grade is 9. The informat grp. does not include 9 in its informat ranges. The numeric value of 9 was not changed to another value since the value of 9 is compatible with the numeric variable type of grade.

```
data convert;
input grade grp.;
cards;
9
;
```

In the following example, an error will occur because Z (which is not specified in the range of format three.) is a character value and the resulting variable grade is numeric.

```
data convert;
input grade three.;
cards;
Z
;
```

This generates the following error message:

```
NOTE: Invalid data for GRADE in line 513 1-2.
RULE:-----1-----2-----3-----4---
513  Z
GRADE=. _ERROR_=1 _N_=1
```

In the following example, the value of character variable grade will be Z for the first observation because Z is a character value. For the second observation, the value of grade will be 1 because the value of 1 does not conflict with the character value type for grade.

```
data convert;
input grade $words.;
cards;
Z
1
;
```

These examples show how important it is to know what the valid values are for a variable. The previous examples show how values that are not listed in the informat ranges can become actual variable values. Variables can acquire invalid values this way. PROC FORMAT has some keywords that can be used for simple data validation.

SIMPLE DATA VALIDATION

An informat can be used to perform simple data validation. If all the valid values for a variable are specified, *all other* values can be considered invalid. The keyword OTHER is used to indicate range values that are not included in all the other ranges for an informat. When _ERROR_ is specified as an informatted value, all values in the corresponding informat range are not valid and a missing value will be assigned to the variable. When _SAME_ is specified as an informatted value, a value in the corresponding informat range stays the same.

Suppose that values from a variable with integer and non-integer values need to be validated and the only valid values are 1 and 2. The following INVALUE statement uses _SAME_, _ERROR_, and OTHER for this task:

```
invalue check 1=_same_
              2=_same_
              other=_error_;
```

An informat's range can be specified as a list of values separated by commas. The following statement is functionally equivalent to the previous one:

```
invalue check 1,2=_same_
              other=_error_;
```

ENHANCED NUMERIC INFORMATS

It is possible to handle character and numeric values in the same informat:

```
proc format;
invalue mixed 0-69=0
              70-74=1
              75-79=1.5
              80-84=2
              85-88=2.5
              89-92=3
              93-96=3.5
              97-100=4
              'excellent'=4;
```

Consider the numbers from 1 to 100. Their increasing numeric order is 1, 2, ..., 99, 100. On the other hand, *character* values from 1 to 100 arranged in increasing *lexicographic* (character value) order result in the following:

```
1, 10, 100, 11, 12, 13, 14, 15, 16, 17, 18, 19
2, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29
.
.
9, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99
```

Unquoted numbers in an informat range are treated as numeric values in SAS release 6.07 or higher. In earlier releases, they were treated as character values and not as numeric values. A value of 9 would be in the 89-92 range for earlier releases instead of the 0-69 range for more recent releases.

SOME SYNTAX NOTES FOR FORMATS

Character format names begin with a dollar sign (\$) while numeric format names do not. A *character format is used to format the values of a character variable. A numeric format is used to format the values of a numeric variable.*

```
proc format;
value $rating 'A'=excellent 'B+'=very good
              'B'=good      'C+'=above average
              'C'=average    'D+'=below average
              'D'=poor       'F'=failure;
value $words  'A'=excellent' 'B+'=very good'
              'B'=good       'C+'=above average'
              'C'=average     'D+'=below average'
              'D'=poor       'F'=failure';
value $one    'A'=4          'B+'=3.5
              'B'=3          'C+'=2.5
              'C'=2          'D+'=1.5
              'D'=1          'F'=0;
value $two    'A'=4          'B+'=3.5
              'B'=3          'C+'=2.5
              'C'=2          'D+'=1.5
              'D'=1          'F'=0;
value letter  4='A'          3.5='B+'
              3='B'          2.5='C+'
              2='C'          1.5='D+'
              1='D'          0='F';
value group   2-high=1       low-1.5=0;
picture grade low-high='9.9' (multiplier=1);
```

For character format \$rating, the formatted values were not enclosed in quotes, but SAS considers these values as enclosed in quotes when dealing with character formats. Hence, format \$rating. and \$words. are equivalent and so are \$one. and \$two.. Formats letter. and group. are numeric formats. Format grade. is a picture format.

In creating user-defined formats, it may become apparent that some formats are similar to others and that certain formats are parts of other formats. That is, some formats appear to be nested in others.

NESTED FORMATS

SAS allows formats to be nested. That is, existing formats (standard or user-defined) can be used as a part of another format. Consider the following formats that have overlapping specifications.

```
proc format;
  value one
    3-3.5='good or very good'
    2-2.5='average or above average'
    1-1.5='poor or below average';
  value two
    4='excellent'
    3-3.5='good or very good'
    2-2.5='average or above average'
    1-1.5='poor or below average'
    0='failure';
```

The specifications in format one. form a subset of the specifications in format two.. Format one. can be nested in format two. and the above code can be rewritten as follows:

```
proc format;
  value one
    3-3.5='good or very good'
    2-2.5='average or above average'
    1-1.5='poor or below average';
  value two
    4='excellent'
    1-3.5=(|one.|)
    0='failure';
```

The above statements will yield a warning message for format two.:

```
WARNING: The ONE (in)format was specified on the
right-hand side of an equals sign, but without a
length specification. PROC FORMAT will assume a
default length of at least 40 for the format being
generated. If this is an insufficient width, you
can rerun PROC FORMAT with an explicit width for
the ONE (in)format, or provide a sufficient
DEFAULT= option.
```

This warning is a reminder that the (in)formatted values have a width associated with them. In format two, the width of the longest formatted value ('average or above average') is 24. The nested format can be rewritten with an explicit width of 24 as follows:

```
value two
  4='excellent'
  1-3.5=(|one24.|)
  0='failure';
```

NUMERIC FORMATS: PICTURE OR VALUE?

The two numeric formats below look very similar. One is specified using a PICTURE statement while a VALUE statement specifies the other. If the VALUE statement is accidentally used when the PICTURE statement should really be used, then all the values that are formatted using format two. will appear as 9.9.

```
proc format;
  picture one    low-high='9.9';
  value two     low-high='9.9';
```

The VALUE statement is used when a numeric value is to be printed as a label that does not necessarily contain numeric characters. The PICTURE statement is used when a numeric value is to be printed in a certain numeric style. For example, numbers considered as monetary values are to be printed in currency style/format.

CREATING NEW VARIABLES FROM EXISTING VARIABLES (RECODING)

For the purpose of providing examples of how formats and informats are used for recoding, the following discussion will focus on the INPUT and PUT functions.

The INPUT function returns the value produced when an expression (*source*) is read using a specified informat. The informat type determines whether the result of the INPUT function is numeric or character. The INPUT function is also used to convert character values to numeric values. Its general syntax without optional arguments is:

INPUT(*source*,*informat*)

The PUT function returns a value using a specified format. It writes the values of a numeric or character variable/constant (*source*) using the specified format. The format must be of the same type as the *source*. The PUT function is also used to convert numeric values to character values and always returns a character value. Its general syntax without optional arguments is:

PUT(*source*,*format*)

For example, the data are letter grades given to students in a class. There is a need to convert the letter grades to numeric grades in order to compute the average grade in that class. The following statements can be added to a DATA step to create a new variable called numgrade.

```
if grade='A' then numgrade=4;
else if grade='B+' then numgrade=3.5;
else if grade='B' then numgrade=3;
else if grade='C+' then numgrade=2.5;
else if grade='C' then numgrade=2;
else if grade='D+' then numgrade=1.5;
else if grade='D' then numgrade=1;
else if grade='F' then numgrade=0;
```

As an alternative, the INPUT function and an informat can be used to achieve the same thing (i.e. create variable numgrade) below. The example below also uses the PUT function and the format \$words. to create variable text..

```
proc format;
  invaluenumber
    'A'=4      'B+'=3.5
    'B'=3      'C+'=2.5
    'C'=2      'D+'=1.5
    'D'=1      'F'=0;
  value $words
    'A'='excellent'
    'B+'='very good'
    'B'='good'
    'C+'='above average'
    'C'='average'
    'D+'='below average'
    'D'='poor'
    'F'='failure';
```

```
data grades;
  input grade $;
  numgrade=input(grade,number.);
  text=put(grade,$words.);
cards;
A
;

PROC PRINT FOR DATA SET GRADES
OBS   GRADE   NUMGRADE   TEXT
1      A         4      excellent
```

FORMATTING VARIABLE OUTPUT

Variables may contain a set of values in a data set, but another set of values needs to appear in the output. Formats control the appearance of data in output. For example, the data are students' numeric grades and these need to be printed in letter grade format. One way to do this is to create another variable that has the letter grades and print this out. Another way is to use a SAS format.

```
proc format;
  value letter 4='A'      3.5='B+'  3='B'
              2.5='C+'  2='C'      1.5='D+'
              1='D'      0='F';
proc print; var grade; format grade letter.;
```

The use of formats is not limited to PROC PRINT. If a frequency table is needed for categories of students (e.g., low and high), using a format can achieve this.

```
**METHOD 1: USES A FORMAT;
proc format;
  value group 2-4='high' 0-1.5='low';
proc freq; tables grade; format grade group.;

**METHOD 2: DOES NOT USE A FORMAT;
data grades; set grades;
  if 2<=grade<=4 then group='high';
  else if 0<=grade<=1.5 then group='low';
proc freq; tables group;
```

In preceding examples, VALUE, INVALUE, and PICTURE statements defined formats/informats. Typing these statements is tedious for very long formats/informats. PROC FORMAT's CNTLIN= option enables PROC FORMAT to accept an input control data set as input for creating formats/informats instead of creating them through VALUE, INVALUE, or PICTURE statements.

INPUT CONTROL DATA SETS (CNTLIN=)

A control data set is a SAS data set that can be used to construct formats and informats with PROC FORMAT. Use PROC FORMAT'S CNTLIN= option to define a format using the data from an input ("IN") control ("CNTL") data set. It is an alternative to typing all the format/informat ranges and formatted/informatted values in VALUE, INVALUE, and PICTURE statements. The CNTLIN= option involves specifying a SAS data set from which the format will be created. The input control data set must have the following variables in it: START, LABEL, and FMTNAME.

- ♦ The START variable contains values in the format/informat ranges in VALUE, INVALUE, and PICTURE statements. There must be no duplicates in the values of the START variable.
- ♦ The LABEL variable contains the formatted/informatted values in VALUE, INVALUE, and PICTURE statements. These are the labels associated with each value of the START variable.
- ♦ The FMTNAME is the name of the format/informat that PROC FORMAT's CNTLIN= option will create.

With these three requirements in mind, consider the following example of an input control data set:

```
data students;
input ssn 1-9 name $ 11-29;
fmtname='names';
rename ssn=start name=label;
cards;
123456789 Aguinaldo, Emilio
987654321 Rizal, Jose
;
```

```
proc format cntlin=students;
```

The CNTLIN= option creates a numeric format called names. by using data set students as an input control data set. The format range consists of social security numbers (ssn) and the formatted values will be the names (name). The above statements are functionally equivalent to the following statements:

```
proc format;
  value names 123456789='Aguinaldo, Emilio'
            987654321='Rizal, Jose';
```

Although the preceding example only shows a format with two values, the CNTLIN= option can generate a format with thousands of values.

The CNTLIN= option will also not accept any duplicates in the START variable. One way to eliminate duplicates is to use the NODUPKEY option with PROC SORT:

```
proc sort data=students
  out=unique
  nodupkey;
by start;
```

Data set students will be sorted by the START variable (ssn) and the results will be placed in output data set unique. Since the NODUPKEY option was specified, the output data set will not have any duplicate ssn values.

LOOK-UP TABLES

Using an input control data set with the CNTLIN= option is a convenient way of creating a lookup table. Suppose that the lookup table consists of social security numbers (ssn). In the example below, the LABEL variable value is set to 'target' (although it may be set to any arbitrary constant):

```
data students;
input ssn 1-9;
fmtname='lookup';
label='target';
rename ssn=start;
cards;
123456789
987654321
;

proc format cntlin=students;
```

The following statements are functionally equivalent to the previous ones:

```
proc format;
value lookup 123456789='target'
987654321='target';
```

Suppose that there is another data set (other) that has social security number (ssn) values. These values need to be checked against the values in the lookup table (format lookup.). Use the following statements to isolate the records in data set other that have ssn values in the lookup table:

```
data matches; set other;
if put(ssn, lookup.)='target';
```

OUTPUT CONTROL DATA SETS (CNTLOUT=)

An output ("OUT") control ("CNTL") data set is created when the CNTLOUT= option is used with the PROC FORMAT statement. The output data set will contain the specifications of the formats and informats created under that PROC FORMAT statement. PROC FORMAT can use the information to create the same or other formats/informats. For example, the data set outform contains the specifications for format group..

```
proc format cntlout=outform;
value $gender 'M'='male' 'F'='female';

proc print data=outform;
title 'Output Control Data Set: outform';

proc contents data=outform;
```

The PROC PRINT and PROC CONTENTS statements produce the following output:

```
Output Control Data Set: outform                                1

      F      D      L      P      N      S      E
      M      E      F      R      O      E      E
      T      F      E      F      T      T      E
      N      A      N      E      F      E      E
      A      A      U      F      I      Y      X
      0      A      G      U      I      D      X
      B      M      L      T      L      I      X
      S      E      T      H      T      P      C
      E      D      Z      X      L      C      L
      O

1 GENDER F F female 1 40 6 6 0 0 0 C N N
2 GENDER M M male 1 40 6 6 0 0 0 C N N
```

Output Control Data Set: outform

2

CONTENTS PROCEDURE

```
Data Set Name: WORK.OUTFORM      Observations:      2
Member Type:   DATA              Variables:       17
Engine:        V612               Indexes:         0
Created:       22:16 Tue, Jul 13, 1999 Observation Length: 63
Last Modified: 22:16 Tue, Jul 13, 1999 Deleted Observations: 0
Protection:                               Compressed:    NO
Data Set Type:                               Sorted:       NO
Label:
```

-----Engine/Host Dependent Information-----

```
Data Set Page Size:      8192
Number of Data Set Pages: 1
File Format:             607
First Data Page:        1
Max Obs per Page:       129
Obs in First Data Page: 2
```

-----Alphabetic List of Variables and Attributes-----

#	Variable	Type	Len	Pos	Label
7	DEFAULT	Num	3	22	Default length
16	EEXCL	Char	1	52	End exclusion
3	END	Char	1	9	Ending value for format
12	FILL	Char	1	46	Fill character
1	FMTNAME	Char	8	0	Format name
9	FUZZ	Num	8	28	Fuzz value
17	HLO	Char	10	53	Additional information
4	LABEL	Char	6	10	Format value label
8	LENGTH	Num	3	25	Format length
6	MAX	Num	3	19	Maximum length
5	MIN	Num	3	16	Minimum length
11	MULT	Num	8	38	Multiplier
13	NOEDIT	Num	3	47	Is picture string noedit?
10	PREFIX	Char	2	36	Prefix characters
15	SEXCL	Char	1	51	Start exclusion
2	START	Char	1	8	Starting value for format
14	TYPE	Char	1	50	Type of format

The labels of the variables in the control output data set describe what specifications are in the data set. The FMTNAME (format name), START (starting value for format), and LABEL (format value label) variables that are required for an input control data set also appear in the output control data set. This data set provides exact information about how PROC FORMAT statements were interpreted.

What are the uses of an output control data set? The output control data set contains all the format/informat specifications of user-defined formats/informats. It is possible to manipulate or change the specifications using programming statements instead of making changes manually. The option to revise the data set via programming is particularly beneficial when dealing with very large formats/informats. Once the output control data set is revised, it can be used as an input control data set to create new formats/informats.

THE NEED FOR ALL VALUES

The output control data set can be a source of all the possible values of interest for a certain variable. For example, a complete list of schools is available through an output control data set because there is a format that involves all the schools. This complete list can be used to check against other data to determine which schools are or are not represented in the other data.

Another application that uses the output control data set involves forcing all known categories of a class variable to appear in output even when all the categories do not occur in the data. For example, the average course grade has to be computed for males and females. However, the data only has data for females.

```
DATA STUDENT;
INPUT GENDER $ GRADE;
CARDS;
F 1
F 2
;

PROC SUMMARY DATA=STUDENT PRINT; CLASS GENDER; VAR GRADE;
TITLE 'PROC SUMMARY FOR DATA SET STUDENT';
```

The PROC SUMMARY output follows:

```
PROC SUMMARY FOR DATA SET STUDENT

Analysis Variable : GRADE
```

GENDER	N	Obs	N	Mean	Std Dev	Minimum	Maximum
F	2	2		1.5000000	0.7071068	1.0000000	2.0000000

In order for the M category to appear in the output, data set FIX is created. This contains all the categories of interest in the CLASS variable gender.

```
DATA FIX;
INPUT GENDER $;
CARDS;
M
F
;

PROC PRINT FOR DATA SET FIX
```

OBS	GENDER
1	M
2	F

Data set FIX is appended to data set STUDENT to form data set PADDED.

```
DATA PADDED; SET STUDENT FIX;

PROC PRINT FOR DATA SET PADDED
```

OBS	GENDER	GRADE
1	F	1
2	F	2
3	M	.
4	F	.

```
PROC SUMMARY DATA=PADDED PRINT; CLASS GENDER; VAR GRADE;
TITLE 'PROC SUMMARY FOR DATA SET PADDED';
```

The PROC SUMMARY output contains category M.

```
PROC SUMMARY FOR DATA SET PADDED

Analysis Variable : GRADE
```

GENDER	N	Obs	N	Mean	Std Dev	Minimum	Maximum
F	3	2		1.5000000	0.7071068	1.0000000	2.0000000
M	1	0		.	.	.	.

If it were always so easy to create the data set with all the categories of interest, then this solution is quite alright. If the categories of interest are available through a format, another solution would be to use the CNTLOUT= option.

The format \$gender . contains the categories of interest. These categorical values can be accessed through the output control data set via the CNTLOUT= option.

```
PROC FORMAT CNTLOUT=OUTFORM;
VALUE $GENDER 'M'='MALE' 'F'='FEMALE';
VALUE $OTHER 'A'='YES' 'B'='NO';
```

For the purpose of providing an example of a control data set that involves specifications for more than one format, the format \$other . was included in the PROC FORMAT statement above.

```
PROC PRINT FOR DATA SET OUTFORM
```

	F	M	A	B	F	M	A	B	F	M	A	B	F	M	A	B	F	M	A	B
1	GENDER	F	F	FEMALE	1	40	6	6	0	0	0	0	0	C	N	N				
2	GENDER	M	M	MALE	1	40	6	6	0	0	0	0	C	N	N					
3	OTHER	A	A	YES	1	40	3	3	0	0	0	0	C	N	N					
4	OTHER	B	B	NO	1	40	3	3	0	0	0	0	C	N	N					

Data set OUTFORM is used to obtain the categories of interest: FEMALE and MALE in the LABEL variable column. The LABEL variable is renamed to CLASSVAR.

```
DATA FIX2;
SET OUTFORM;
IF FMTNAME='GENDER';
KEEP LABEL;
RENAME LABEL=CLASSVAR;
```

```
PROC PRINT FOR DATA SET FIX2
```

OBS	CLASSVAR
1	FEMALE
2	MALE

The PUT function created the CLASSVAR variable for data set STUDENT2.

```
DATA STUDENT2;
INPUT GENDER $ GRADE;
CLASSVAR=PUT(GENDER,$GENDER.);
CARDS;
F 1
F 2
;
```

```
PROC PRINT FOR DATA SET STUDENT2
```

OBS	GENDER	GRADE	CLASSVAR
1	F	1	FEMALE
2	F	2	FEMALE

Data set FIX2 is appended to data set STUDENT2 to form data set PADDED2.

```
DATA PADDED2;
SET STUDENT2 FIX2;
```

```
PROC PRINT FOR DATA SET PADDED2
```

OBS	GENDER	GRADE	CLASSVAR
1	F	1	FEMALE
2	F	2	FEMALE
3	.	.	FEMALE
4	.	.	MALE

```
PROC SUMMARY DATA=PADDED2 PRINT; CLASS CLASSVAR; VAR GRADE;
TITLE 'PROC SUMMARY FOR DATA SET PADDED2';
RUN;
```

The PROC SUMMARY output follows:

```
PROC SUMMARY FOR DATA SET PADDED2
Analysis Variable : GRADE

CLASSVAR N Obs N      Mean      Std Dev      Minimum      Maximum
-----
FEMALE      3  2      1.5000000      0.7071068      1.0000000      2.0000000
MALE          1  0
```

This paper is a sampler of basic programming situations that can be simplified by using SAS formats/informats. An important aspect of formats/informats that is not discussed in this paper is formats and informats can be stored in SAS catalogs and retrieved for later use.

CONCLUSION

SAS formats/informats can be used in a variety of programming situations. Taking advantage of their versatility and convenience is a good programming habit.

REFERENCES

Mason, P. (1996). *In the Know...SAS Tips & Techniques From Around the Globe*. Cary, NC: SAS Institute, Inc.

Patton, N. (1998). *IN & OUT of CNTL with PROC FORMAT*. Proceedings of the Sixth Annual Conference of the Southeast SAS Users Group, Norfolk, VA, USA, 6, 269-270.

SAS Institute Inc. (1990), *SAS Language Reference, Version 6, First Edition*, Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1990), *SAS Procedures Guide, Version 6, Third Edition*, Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1990), *SAS Programming Tips: A Guide to Efficient SAS Processing*, Cary, NC: SAS Institute, Inc.

SAS Institute Inc. (1996), *SAS Technical Report P-222: Changes and Enhancements to Base SAS Software*, Cary, NC: SAS Institute, Inc.

TRADEMARK NOTICE

SAS is a registered trademark or trademark of the SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Imelda C. Go
Office of Research and Evaluation
Richland County School District One
1616 Richland St.
Columbia, SC 29201

Tel.: (803) 733-6079
Fax: (803) 929-3873
icgo@juno.com

