

## Beyond FORMAT Basics

Mike Zdeb, University@Albany School of Public Health, Rensselaer, NY

### ABSTRACT

Beginning and even intermediate level SAS® users sometimes never venture beyond the basics in using formats. They may be content with writing formats for substituting labels for the values of variables or for grouping observations based on the values of variables. This paper goes beyond these basics and shows you: how to create single and multiple formats from SAS data sets with CNTLIN data sets; when to use the PUTN and PUTC functions as alternatives to the PUT function; how to use formats as an alternative to data step matched-merges and PROC SQL when selecting records from a file or when combining information from multiple files or data sets; how to create nested (or recursive) formats; how to use pre-loaded formats to force zero count value ranges into procedure results; how to create and use multi-label formats (formats with overlapping value ranges); how to create user-written informats to clean data as it is read from raw data files. The material is intended for users who are new to SAS and who want to learn how to expand their use of the formats.

### INTRODUCTION

Though this paper is intended to take you beyond the basics, a few basics are reviewed first. What are formats and informats? According to SAS documentation...*A format is an instruction that SAS uses to write data values. You use formats to control the written appearance of data values, or, in some cases, to group data values together for analysis...An informat is an instruction that SAS uses to read data values into a variable.*

SAS documentation refers to six format and eight informat categories. However, the listed categories are bit confusing in that they include two categories that are also referred to as types, i.e. character and numeric. For purposes of this paper, think of formats and informats as being either SAS-supplied or user-written and within each of those categories, there are numeric and character formats and informats.

User-written formats and informats are created with PROC FORMAT. There are a number of rules for creating and using formats. One of the rules forces you to think ahead in that when you create a format in that the format type determines the variable type with which the format (informat) can be used. The following rules (not meant to be an exhaustive list) apply to both formats and informats except where noted...

#### FORMAT TYPES

- if you plan to use a format with a character variable, create a character format
- if you plan to use a format with a numeric variable, create a numeric format

#### FORMAT NAMES

- the name of the format determines the format type
- if a format name begins with a \$, it is character format
- if there is no \$ at the start of the name, it is a numeric format
- a format name in V9 can be up to 32 characters in length (a \$ counts as part of the 32 characters)
- NOTE: informat names in V9 can be up to 31 characters in length (a \$ counts as part of the 31 characters)
- a format name must start with a letter or underscore (the first character or the first character after the \$)
- a format name can only contain letters, numbers, and underscores
- a format name cannot end in a number

There are also rules that apply to constructing user-written formats and informats and these will be referred to in various parts of the following sections. One last item to mention before venturing beyond the basics is that many of the examples use data from the field of public health, reflecting the data most familiar to the author.

### CNTLIN DATA SETS

One common use of a format in SAS is to assign labels to values of a variable. The rules for creating a format with PROC FORMAT are quite simple, the most important being to think ahead, make sure that format type matches the variable type --- create character formats for character variables and numeric formats for numeric variables. When the list of variable values is short, it is easy to type all the values and their labels.

```
* EXAMPLE 1 - write your own CHARACTER format;
data test;
input id : $5. drg : $3. @@;

datalines;
12345 001 34567 005 67890 310 99999 002 87654 004
;
run;
```

```

proc format;
value $drg ❶
'001' = 'CRANIOTOMY AGE >17 EXCEPT FOR TRAUMA'
'002' = 'CRANIOTOMY FOR TRAUMA AGE >17'
'004' = 'SPINAL PROCEDURES'
'005' = 'EXTRACRANIAL VASCULAR PROCEDURES'
other = 'UNKNOWN'; ❷
;
run;

title 'EXAMPLE 1 - FORMAT CREATED BY KEYING VALUES AND LABELS';
proc report data=test nowd;
columns id drg drg=drglabel; ❸
define drglabel / format=$drg.; ❹
run;

```

EXAMPLE 1 - FORMAT CREATED BY KEYING VALUES AND LABELS ❺

id	drg	drg
12345	001	CRANIOTOMY AGE >17 EXCEPT FOR TRAUMA
34567	005	EXTRACRANIAL VASCULAR PROCEDURES
67890	310	UNKNOWN
99999	002	CRANIOTOMY FOR TRAUMA AGE >17
87654	004	SPINAL PROCEDURES

The data set TEST contains two variables, an ID number and a DRG (short definition: a 3-digit number that summarizes what happened to you during a hospital stay). You want to create a report that shows both the DRG and a literal description of the DRG, so you create a format. The character format \$drg is to be used with the character variable DRG ❶. An OTHER condition is specified to assign a label of UNKNOWN to variable values not found in the list codes shown to the left of the equals in PROC FORMAT ❷. PROC REPORT is used to print both the variable value and its label, using an alias to display the variable DRG twice ❸. The format \$DRG is assigned to the alias ❹. The results of PROC REPORT show the variable values and labels ❺.

If the number of variable values is large and the values and their labels are available in some computer accessible mode (flat file, spread sheet, data base), there is no need to retype the list. A data set can be created that contains all the information that PROC FORMAT needs to create a format. The data set must contain at a minimum three variables: FMTNAME, START, LABEL. Once created, this data set is used as a CNTLIN (control in) data set by PROC FORMAT to create a format. Assume that you have a flat file with values of DRGs and their labels. There are over 600 entries in the file and an excerpt looks as follows....

```

057 T&A PROC,EXCEPT TONSILLECTOMY &/OR ADENOIDECT ONLY,AGE >17
058 T&A PROC,EXCEPT TONSILLECTOMY &/OR ADENOIDECT ONLY,AGE <18
059 TONSILLECTOMY &/OR ADENOIDECTOMY ONLY, AGE >17
060 TONSILLECTOMY &/OR ADENOIDECTOMY ONLY, AGE <18
061 MYRINGOTOMY W TUBE INSERTION AGE >17
062 MYRINGOTOMY W TUBE INSERTION AGE <18

```

\* EXAMPLE 2 - create a CHARACTER format with a CNTLIN data set;

```

data drg_fmt / view=drg_fmt; ❶
retain fmtname '$drgnew'; ❷
infile 'j:\epi697\data\drgs.dat' trunccover;
input start $3. +1 label $50.; ❸
run;

proc format cntlin=drg_fmt; ❹
run;

data test;
input id : $5. drg : $3. @@;

datalines;
12345 001 34567 005 67890 310 99999 002 87654 004 44444 062 55555 059 12121 888
;
run;

title 'EXAMPLE 2 - FORMAT CREATED WITH A CNTLIN DATA SET';
proc report data=test nowd;
columns id drg drg=drglabel;
define drglabel / format=$drgnew.;
run;

```

**EXAMPLE 2 - FORMAT CREATED WITH A CNTLIN DATA SET ❶**

```

id      drg      drg
12345   001      CRANIOTOMY AGE >17 EXCEPT FOR TRAUMA
34567   005      EXTRACRANIAL VASCULAR PROCEDURES
67890   310      TRANSURETHRAL PROCEDURES W CC
99999   002      CRANIOTOMY FOR TRAUMA AGE >17
87654   004      SPINAL PROCEDURES
44444   062      MYRINGOTOMY W TUBE INSERTION AGE <18
55555   059      TONSILLECTOMY &/OR ADENOIDECTOMY ONLY, AGE >17
12121   888      888

```

The CNTLIN data set is only needed to create the format. Therefore, a data set view is created instead of a data set ❶. The first of the minimum set of variables is added to the data set using a RETAIN statement ❷. The variable FMTNAME takes the role of the keyword VALUE in PROC FORMAT. The values of the variables and their labels are stored in a file that has variable length records. The other two variables in the minimum data set are read from the flat file. The variable values (the entries on the left side of the '=' in the value statement) are given the name START, while the variable labels (the entries on the right side of the '=' in the value statement) are given the name LABEL ❸. The data set view is used with the CNTLIN= option of PROC FORMAT to create the format \$DRGNEW ❹. The SAS LOG shows the following...

```

9      proc format cntlin=drg_fmt;
NOTE: The infile "j:\epi697\data\drgs.dat" is:
      File Name=j:\epi697\data\drgs.dat,
      RECFM=V,LRECL=256

```

```

NOTE: Format $DRGNEW has been output.
10     run;

```

One note shows that the format \$DRGNEW was created, the same message that appears when all the text for a format is used in PROC FORMAT as in EXAMPLE1. If you have never used a SAS view, seeing a note in the LOG about an external data file being read might look out of place. The data needed for the CNTLIN data set DRG\_FMT is read when the PROC is run, not when the view is created in the previous program step. The output from PROC REPORT shows the variable values and labels, except for the value 888 that was not found in the format ❺. Since no OTHER condition was specified as in EXAMPLE 1, the unformatted value of the variable was printed.

Specification of an OTHER condition requires adding one more variable (HLO) to the CNTLIN data set. Just as the three variables in the minimum CNTLIN data set were recognized by PROC FORMAT, a variable named HLO is also recognized and can be used to create either a H(igh), L(ow), or O(ther) condition by assigning the variable HLO a value of H, L, or O respectively.

**\* EXAMPLE 3 - create a CHARACTER format with a CNTLIN data set (HLO variable added);**

```

data drg_fmt / view=drg_fmt;
length label $50; ❶
retain fmtname '$drghlo';
if last then do; ❷
    hlo = 'o'; ❸
    label = 'UNKNOWN'; ❹
    output; ❺
end;
infile 'j:\epi697\data\drgs.dat' trunccover end=last;
input start $3. +1 label $50.;
output; ❻
run;

proc format cntlin=drg_fmt;
run;

data test;
input id : $5. drg : $3. @@;
datalines;
12345 001 34567 005 67890 310 99999 002 87654 004 44444 062 55555 059 12121 888
;
run;

title 'EXAMPLE 3 - FORMAT CREATED WITH A CNTLIN DATA SET (HLO VARIABLE ADDED)';
proc report data=test nowd;
columns id drg drg=drghlo;
define drghlo / format=$drghlo.;
run;

```

**EXAMPLE 3 - FORMAT CREATED WITH A CNTLIN DATA SET (HLO VARIABLE ADDED) ⑦**

```

id      drg   drg
12345   001   CRANIOTOMY AGE >17 EXCEPT FOR TRAUMA
34567   005   EXTRACRANIAL VASCULAR PROCEDURES
67890   310   TRANSURETHRAL PROCEDURES W CC
99999   002   CRANIOTOMY FOR TRAUMA AGE >17
87654   004   SPINAL PROCEDURES
44444   062   MYRINGOTOMY W TUBE INSERTION AGE <18
55555   059   TONSILLECTOMY &/OR ADENOIDECTOMY ONLY, AGE >17
12121   888   UNKNOWN

```

A length of 50 is assigned to the character variable LABEL ①. There are a few statements in this data step that might be good tests of your understanding of how a data step executes. (*Why is the length statement needed?*) An IF-THEN statement is used to add the variable HLO to the data set after the last value and its label have been read from the flat file and added to the data set ②. (*Why does this work even though the statement appears before the input statement? Why might this be a good place to always put a check for having read the last observation of a file or data set?*) The variable HLO is assigned a value of 'o' to set up an OTHER condition in the format ③. The label for the other condition is 'UNKNOWN' ④. An output statement is used to add the other condition to the data set ⑤ (*Why is the output statement needed? Why is this output statement ⑥ also needed?*) The output from PROC REPORT shows that the OTHER condition was successfully added to the format ⑦.

Each DRG has an associated set of limits on length of stay in a hospital, a minimum and maximum allowable number of days. Imagine that you have a large file of records that each have a DRG and length of stay. You also have a file in which each record has a DRG plus the minimum and maximum stay. Your task is to use the allowable limits for each DRG to find records in the large file with a length of stay that fall outside the allowable range. You could create SAS data sets, merge the two files on DRG, then select those observations with a length of stay that is out-of-range. Another more efficient method is to use a format.

**\* EXAMPLE 4 - USE A FORMAT TO SELECT RECORDS FROM A LARGE FILE;**

```

proc format; ①
value _370_ 2-10 = '1'   other = '0';
value _372_ 2-6  = '1'   other = '0';
run;

```

```

data moms;
format ssn ssn.;
input ssn : commall1. drg : $3. los; ②
fmt = cat('_',drg,'_'); ③
if putn(los,fmt) eq '0'; ④
drop fmt;
datalines;
001-36-1234 370 15
089-45-0222 372 2
345-67-8901 370 1
123-99-6789 372 25
;
run;

```

```

title 'EXAMPLE 4 - UNACCEPTABLE STAYS';
proc print data=moms;
run;

```

**EXAMPLE 4 - UNACCEPTABLE STAYS ⑤**

Obs	ssn	drg	los
1	001-36-1234	370	15
2	345-67-8901	370	1
3	123-99-6789	372	25

Two formats are created and the name of the format for each DRG is the DRG with an underscore as a prefix and suffix (remember, a format name must start with a letter or underscore and cannot in a number) ①. DRG 370 has a minimum allowable stay of 2 days and a maximum of 10, while for DRG 372 the range is 2 to 6. Records are read that contain a DRG plus the length of stay for various hospital stays ②. The CAT (concatenate) function is used to create the variable FMT, adding an underscore as a prefix and suffix to the value if the DRG ③. Notice that the value of the variable FMT has the same structure as the names of the formats created in the previous job step. The PUTN function is used to compare the value of the variable LOS to the acceptable range of length of stay that was previously stored in a format ④. You might be familiar with the PUT function that can be used in a similar manner, but a PUT function requires a format name as the second argument, not a variable name or an expression that resolves to a variable name. In this example, the format name must change as the DRG

changes and the PUTN function allow this (more about PUTN and PUTC in a subsequent section). The output from PROC PRINT shows that the correct records are selected from the data file.

As with EXAMPLE1, keying all the values of the format is not tedious when there is a small number of DRGs and acceptable ranges of length of stay. However, when the list is large, the same approach can be used as in EXAMPLE2, use a CNTLIN data set. More than one format can be created using one CNTLIN data set. Whenever the content of the variable FMTNAME changes in the CNTLIN data set, PROC FORMAT assumes that a new format is to be created.

```
* EXAMPLE 5 - creating many formats with one CNTLIN data set;
data manyfmts;
infile "j:\epi697\data\acceptable_stays.txt"; ❶
input drg : $3. start end; ❷

fmtname = cat('_',drg,'_'); ❸
label = '1';
output; ❹
hlo = 'o'; ❺
label = '0';
output;
run;

title 'EXAMPLE 5 - PORTION OF CNTLIN DATA SET';
proc print data=manyfmts (obs=10);
run;

title 'EXAMPLE 5 - TWO FORMATS CREATED USING A CNTLIN DATA SET';
proc format cntlin=manyfmts;
select _001_ _002_; ❻
run;
```

EXAMPLE 5 - PORTION OF CNTLIN DATA SET ❼

Obs	drg	start	end	fmtname	label	hlo
1	001	3	51	_001_	1	
2	001	3	51	_001_	0	o
3	002	2	30	_002_	1	
4	002	2	30	_002_	0	o
5	006	1	4	_006_	1	
6	006	1	4	_006_	0	o
7	007	3	46	_007_	1	
8	007	3	46	_007_	0	o
9	008	2	21	_008_	1	
10	008	2	21	_008_	0	o

EXAMPLE 5 - TWO FORMATS CREATED USING A CNTLIN DATA SET ❽

FORMAT NAME: _001_    LENGTH:    1    NUMBER OF VALUES:    2					
MIN LENGTH:   1   MAX LENGTH: 40   DEFAULT LENGTH   1   FUZZ: STD					
START		END	LABEL   (VER. V7 V8    22JUN2005:10:05:18)		
		3	51	1	
**OTHER**		**OTHER**		0	

  

FORMAT NAME: _002_    LENGTH:    1    NUMBER OF VALUES:    2					
MIN LENGTH:   1   MAX LENGTH: 40   DEFAULT LENGTH   1   FUZZ: STD					
START		END	LABEL   (VER. V7 V8    22JUN2005:10:05:18)		
		2	30	1	
**OTHER**		**OTHER**		0	

A file containing 600+ DRGs and acceptable stay ranges is read ❶. In EXAMPLE2, only a START variable was added to the CNTLIN data set. In this example, a range of values is must be specified in the format so both START and END variables are added to the data set ❷. As in EXAMPLE4, the value if the variable DRG with an underscore prefix and suffix is used for the format name ❸. An OUTPUT statement adds an observation to the CNTLIN data set, an observation that has an acceptable stay range ❹. The next two steps add an OTHER condition (as was done in EXAMPLE2) ❺. Thus, each record in the input

data file results in two observations in the CNTLIN data set. Notice in PROC FORMAT, a new statement is added, SELECT ⑥. In addition to creating a format, PROC FORMAT will display the formats specified in the SELECT statement.

A portion of the CNTLIN data set is printed ⑦. Notice that the value of the variable FMTNAME changes after every two observations. The first two values of FMTNAME are \_001\_ and \_002\_. The results of using the SELECT statement in PROC FORMAT display these two formats ⑧. The SAS LOG displays the names of the formats that can be used as in EXAMPLE4 screen data for unacceptable lengths of stay.

```
22  proc format cntlin=manyfmts;
NOTE: Format _001_ has been output.
NOTE: Format _002_ has been output.
<many more lines>
NOTE: Format _875_ has been output.
NOTE: Format _876_ has been output.
23  select _001_ _002_;
24  run;
```

### PUTN AND PUTC

EXAMPLE3 used the PUTN function with only a short discussion as to how it differs from the PUT function. The PUT function has a variety of uses. One is to convert a numeric value to a character value. For example, a character variable can be created from a numeric variable named ZIP (a ZIP code) as follows...

```
zipc = put(zip,5.);
```

Another use is to create a new variable based on the value of a current variable plus a user-written format...

```
proc format;
value age
low-4 = '<5' 5-9 = '5-9' 10-14='10-14'
other = 'OTHER' . = 'MISSING';
run;
```

```
<then in a data step>
age_group = put(age,age.);
```

In both of the above uses, the name of either a SAS-supplied or user-written format had to be hardcoded as the second argument in the PUT function. As shown in EXAMPLE4, there might be an occasion where the value of the format to be used in a PUT function varies with one's data. In that case, either a PUTN or PUTC function must be used. Both PUTN and PUTC allow the second argument to be either a variable whose value is the name of a format or a SAS expression that resolves to a format name. EXAMPLE4 used a variable whose value is a format name...

```
fmt = cat('_',drg,'_');
if putn(los,fmt) eq '0';
drop fmt;
```

It could also have been written...

```
if putn(los,catt('_',drg,'_')) eq '0';
```

since the second argument resolves to the name of a format (eliminating the need to create, then drop, the new variable FMT). The PUTN function is used since the LOS is a numeric variable and the second argument is a numeric format. The following is another example of using the PUTN function, not to select observations as in EXAMPLE4, but to create a new variable. For each age group (values 2 through 5), there is a different set of ranges to be used to place BMI (body mass index) into one of three groups.

```
* EXAMPLE 6 - USE THE PUTN FUNCTION TO ADD A NEW VARIABLE TO A DATA SET;
proc format; ①
value age2_ low - 17.9 = 'normal' 18.0 - 19.0 = 'risk' 19.1 - high = 'overweight';
value age3_ low - 17.1 = 'normal' 17.2 - 18.1 = 'risk' 18.2 - high = 'overweight';
value age4_ low - 16.7 = 'normal' 16.8 - 17.9 = 'risk' 18.0 - high = 'overweight';
value age5_ low - 16.7 = 'normal' 16.8 - 18.1 = 'risk' 18.2 - high = 'overweight';
run;

data bmi;
input age_group : $1. bmi @@;
bmi_group = putn(bmi,catt('age',age_group,'_')); ②
```

```
datalines;
2 17.8 5 16.7 2 18.6 3 17.6 4 17.0 3 18.2 4 18.0
;
run;

title 'EXAMPLE 6 - BMI_GROUP ADDED USING THE PUTN FUNCTION';
proc print data=bmi;
run;
```

#### EXAMPLE 6 - BMI\_GROUP ADDED USING THE PUTN FUNCTION ③

Obs	age_group	bmi	bmi_group
1	2	17.8	normal
2	5	16.7	normal
3	2	18.6	risk
4	3	17.6	risk
5	4	17.0	risk
6	3	18.2	overweight
7	4	18.0	overweight

Four numeric formats are created, each having a value of the variable AGE\_GROUP embedded in the format name ❶. The variable BMI\_GROUP is created using the PUTN function, with the value of the variable AGE\_GROUP used to choose the appropriate format. ❷. The output from PROC PRINT shows the values of the new variable ❸. When using character variables and selecting among character formats, the PUTC function is used.

#### \* EXAMPLE 7 - USE THE PUTC FUNCTION TO ADD A NEW VARIABLE TO A DATA SET;

```
proc format; ❶
value $FR 'A'-'D' = 'PASS' other = 'FAIL';
value $SO 'A'-'C' = 'PASS' other = 'FAIL';
value $JR 'A'-'B' = 'PASS' other = 'FAIL';
value $SR 'A'      = 'PASS' other = 'FAIL';
run;
```

```
data students;
length pass_fail $4.; ❷
input class : $2. grade : $1. @@;
pass_fail = putc(grade,cat('$',class)); ❸
```

```
datalines;
FR D SO A SO D JR B JR C SR A SR B
;
run;
```

```
title 'EXAMPLE 7 - PASS_FAIL ADDED USING THE PUTC FUNCTION';
proc print data=students;
var class grade pass_fail;
run;
```

#### EXAMPLE 7 - PASS\_FAIL ADDED USING THE PUTC FUNCTION ④

Obs	class	grade	pass_fail
1	FR	D	PASS
2	SO	A	PASS
3	SO	D	FAIL
4	JR	B	PASS
5	JR	C	FAIL
6	SR	A	PASS
7	SR	B	FAIL

Four character formats are created with different ranges of passing grades for each of the formats ❶. A LENGTH statement is REQUIRED to ensure that the value of the new variable PASS\_FAIL is not truncated ❷. The PUTC function is used to create the new variable PASS\_FAIL based on the value of the variable CLASS and the appropriate format ❸. The output from PROC PRINT shows the value of the new variable.

Without the LENGTH statement in the data step, the value of PASS\_FAIL would be either 'P' or 'F', not 'PASS' or 'FAIL'. When the PUTC function is used to create a new variable, the length of the new variable is derived from the length of the old variable used within the PUTC function. Since GRADE has a length of 1, the default length of PASS\_FAIL is 1. The LENGTH statement is used prior to using the PUTC function to prevent truncation of the value of PASS\_FAIL.



Note that this behavior of PUTC is the opposite of the PUT function. When the PUT function is used to create a new variable, the length of the new variable is derived from the format used in the PUT function, not the old variable. If the data step had contained a PUT function with a hard-coded format name, for example...

```
pass_fail = put(grade,$SR.);
```

the length of PASS\_FAIL would be 4 even without the LENGTH statement since the longest label in the format \$SR is four characters.

## RECORD SELECTION

In addition to providing a mechanism to label variable values, formats are also useful in selecting records from a file based on the value of a variable. There are many methods that can be used for record selection (data step merge, IF-THEN statements, PROC SQL). Using a CNTLIN to create a lookup table (a format) of acceptable records provides a very fast alternative to the other methods. Note, this is similar to the technique used in EXAMPLE4 where records were selected based on the values stored in formats.

Imagine that you are given a small file of social security numbers and your task is to extract records from a very large file that have social security numbers corresponding to those in your small file. The first five records from the large data file look as follows...

```
749-36-3208      1
905-36-3570      2
785-36-3980      3
187-36-7761      4
966-36-2636      5
```

while the first five records from the small file of social security numbers are...

```
521-36-9333
223-36-9993
088-36-4600
449-36-3866
812-36-2568
```

```
* EXAMPLE 8 - USE A CNTLIN DATA SET FOR RECORD SELECTION;
data selectssns / view=selectssns; ❶
retain fmtname 'ssnok' label 'ok'; ❷
infile 'j:\epi697\data\getssns.dat';
input start commall.; ❸
run;

proc format cntlin=selectssns; ❹
run;

data goodssns;
infile 'j:\epi697\data\lotassns.dat';
input ssn commall. record $6.;
if put(ssn,ssnok.) eq 'ok'; ❺
run;
```

As in previous examples, a CNTLIN data set is created as a view ❶. A RETAIN statement is used to name the format. Since the same label is used for all observations in the CNTLIN data set, the retain statement is also used to give a value to the variable LABEL, i.e. 'ok' ❷. The social security numbers are read as the variable START from the small data file using a comma informat to get rid of the embedded dashes ❸. PROC FORMAT plus a CNTLIN data set creates the format that will be used to select records from the large data file ❹. The format created by using a CNTLIN data set is used to screen the data and select only those records that match a social security number in the lookup table ❺.

Here is a portion of the LOG from using this SAS code when there were 520 and 10,000 records in the small and large data files respectively...

```
418 data goodssns;
419 infile 'j:\epi697\data\lotassns.dat';
420 input ssn commall. record $6.;
421 if put(ssn,ssnok.) eq 'ok';
422 run;
```



NOTE: The infile 'j:\epi697\data\lotassns.dat' is:  
 File Name=j:\epi697\data\lotassns.dat,  
 RECFM=V,LRECL=256

NOTE: 10000 records were read from the infile 'j:\epi697\data\lotassns.dat'.  
 The minimum record length was 17.  
 The maximum record length was 17.

NOTE: The data set WORK.GOODSSNS has 520 observations and 2 variables.

NOTE: At least one W.D format was too small for the number to be printed. The decimal may be shifted by the "BEST" format.

Look at the last note in the LOG. The note refers to a number being printed using a format that was too small to accommodate the full value of the number. Though it may not be obvious, this message results from the PUT statement, the default length of the format SSNOK, and there being no OTHER condition in the format. Values of the social security number found in the format are converted to 'ok'. Those not found maintain their value from the raw data file and the number contains nine digits. Every time the data step encounters a social security number that is not in the format, the PUT statement results in a 9-digit number trying to fit into a 2-character wide space (the default length of the format SSNOK). If you do not mind the note, you can stop here. If you do not like seeing such notes in your LOG, you can modify the SAS code in the first data step. One way is to add another variable, DEFAULT, to the CNTLIN data set...

```
data selectssns / view=selectssns;
retain fmtname 'ssnok' label 'ok' default 9;
infile 'j:\epi697\data\getssns.dat';
input start commall.;
run;
```

Since the social security numbers contain nine digits, the numeric variable DEFAULT (sets the default length of the format) is given a value of 9. Another way to get rid of the message is to use an HLO variable to add an OTHER condition to the format.

```
* EXAMPLE 9 - USE A CNTLIN DATA SET FOR RECORD SELECTION (ADD AN OTHER CONDITION);
data selectssns / view=selectssns;
retain fmtname 'ssnok' label 'ok';
infile 'j:\epi697\data\getssns.dat' end=last;
if last then do;
  hlo = 'o';
  label = 'no'; ❶
  output;
end;
input start commall.;
output;
run;
```

Now, when a social security number is not found in the format, the PUT statement will return the value 'no' ❶. At the start of this section, using a format for record selection was suggested as an alternative to a data step merge and PROC SQL. One advantage of using a format for record selection is that records can be selected as a raw data file is being read rather than after it has been converted to a SAS data set. Both merge and PROC SQL require SAS data sets. Merge would also require sorting both the small and large data files prior to merging by social security number.

## RECORD MATCHING

In the last example, the lookup table was used to select records. The same technique can be used to add information from one file to that in another. Once again, this is an alternative to a data step merge and PROC SQL. Instead of using a small file with only social security numbers to select records, change the task to one where you not only select records, but also add information from the small file to that in the large file. The large file is the same as used in EXAMPLE7. However, the small file of social security numbers now contains another variable, GENDER and the task is now to select records and add the value of the variable GENDER to the resulting data set. The first five records of the small data file now look as follows...

```
521-36-9333 F
223-36-9993 F
088-36-4600 F
449-36-3866 F
812-36-2568 M
```

```
* EXAMPLE 10 - ADD INFORMATION TO A RECORD USING A CNTLIN DATA SET;
data selectssns / view=selectssns;
retain fmtname 'addone'; ❶
infile 'j:\epi697\data\getssnsa.dat' end=last;
```

```

if last then do;
    hlo = 'o';
    label = 'X'; ❷
    output;
end;
input start : commall. label : $1.; ❸
output;
run;

proc format cntlin=selectssns;
run;

data goodssns;
infile 'j:\epi697\data\lotassns.dat';
input ssn commall. record $6.;
gender = put(ssn,addone.); ❹
if gender ne 'X'; ❺
run;

```

The numeric format ADDONE is created ❶. The other condition in the format (social security numbers not found in the lookup table) will have a label of 'X' ❷. The social security number is the START variable and the gender in the LABEL variable ❸. The variable GENDER is added to the data set using the format and a PUT statement ❹. (*What is the length of the variable GENDER?*) Only those observations that are found in the lookup table are added to the data set ❺.

The label is not limited to one variable. Various techniques can be used to place more information in the label. One of them are shown in the next example. The small data file of social security numbers now contains four variables, those shown in the last example, plus a zip code and a salary. The first five records now look as follows...

```

521-36-9333 F 61485    20494
223-36-9993 F 23911    15940
088-36-4600 F 71653    107478
449-36-3866 F 29477    49120
812-36-2568 F 64741    32370

```

```

* EXAMPLE 11 - ADD SEVERAL VARIABLES TO A DATA SET USING A CNTLIN DATA SET;
data selectssns / view=selectssns;
length label $ 18.;
retain fmtname 'addmany';
infile 'j:\epi697\data\getssnsb.dat' end=last;
if last then do;
    hlo = 'o';
    label = 'X';
    output;
end;
input start : commall. gender : $1. zip : $5. salary : $10.; ❶
label = catx('/',gender,zip,salary); ❷
output;
drop gender zip salary;
run;

proc format cntlin=selectssns;
run;

data goodssns (drop=info);
length gender $1 zip $5;
infile 'j:\epi697\data\lotassns.dat';
input ssn commall. record $6.;
info = put(ssn,addmany.); ❸
if info ne 'X'; ❹
gender = scan(info,1,'/'); ❺
zip     = scan(info,2,'/');
salary = input(scan(info,3,'/'),best.);
format ssn ssn. salary dollar10.;
run;

title 'EXAMPLE 11 - RECORD SELECTION PLUS ADD THREE VARIABLES USING A CNTLIN DATA SET';
proc print data=goodssns (obs=10);
var ssn record gender zip salary;
run;

```

**EXAMPLE 11 - RECORD SELECTION PLUS ADD THREE VARIABLES USING A CNTLIN DATA SET ⑦**

Obs	ssn	record	gender	zip	salary
1	521-36-9333	9	F	61485	\$20,494
2	223-36-9993	27	F	23911	\$15,940
3	088-36-4600	50	F	71653	\$107,478
4	449-36-3866	97	F	29477	\$49,120
5	812-36-2568	139	F	64741	\$32,370
6	601-36-9405	171	M	95730	\$31,910
7	019-36-5423	177	M	15330	\$25,550
8	723-36-2545	179	F	95923	\$63,948
9	737-36-9625	183	F	05905	\$7,872
10	597-36-5475	197	M	59208	\$78,944

All the variables are read as character variables from the small file ①. The variable label is created by concatenating three variables, separating them with a slash ②. The social security number is used to find information in the lookup table ③. A subsetting if statement is used to see if information is found ④. For those records found in the lookup table, the information is separated into variables using the SCAN function ⑤. Since salary should be numeric, the input function to create a numeric variable ⑥. (Notice the LENGTH statement. Without that statement, what is the length of variable gender? of variable zip?) A portion of the new data set is shown using PROC PRINT ⑦.

The first five and last five observations in the CNTLIN data set SELECTSSNS used to create the format SELECTSSNS look as follows...

Obs	label	fmtname	hlo	start
1	F/61485/20494	addmany		521369333
2	F/23911/15940	addmany		223369993
3	F/71653/107478	addmany		88364600
4	F/29477/49120	addmany		449363866
5	F/64741/32370	addmany		812362568
<more observations>				
516	M/21816/7272	addmany		871362060
517	M/88978/74145	addmany		744362308
518	M/74602/24866	addmany		71366042
519	F/31197/5199	addmany		360361304
520	X	addmany	o	.

The variable LABEL has all the data needed to add three variables to the data set GOODSSNS. Since no value is assigned to the variable HLO until after the last observation is read from the small file, its value is missing until the last observation in the data set is created.

**NESTED FORMATS**

A recent addition to PROC FORMAT is the ability to nest formats, i.e. to reference one or more formats from within a given format. This process is sometimes referred to as format recursion. Nesting is possible with both SAS-provided or user-written formats. One example where format nesting is very useful is a situation where you are using a previously created format from a format library. If you need to make a small change or addition to the format for a particular SAS job, there is no need to modify the format library. The next example shows how to make an addition and change to an already present, user-written format. First, a format is created that links a name to a hospital number (to be thought of as a format in a format library). Then, another hospital is added to the lookup table, and the label for one hospital is changed.

```
* EXAMPLE 12 - NESTING FORMATS (USER-WRITTEN);
* ASSUME THIS IS A FORMAT IN A FORMAT LIBRARY;
proc format; ①
value $num2nam
'0001' = 'ALBANY MEDICAL CENTER HOSPITAL'
'0002' = 'CHILDS HOSPITAL1'
'0005' = 'ST PETERS HOSPITAL'
'0012' = 'OSWEGO HOSPITAL'
'0016' = 'CATHOLIC MED CTR'
'0025' = 'COHOES MEMORIAL HOSPITAL'
other = 'UNKNOWN'
;
run;
```

```
* MAKE TWO MODIFICATIONS - ONE ADDITION, ONE LABEL CHANGE';
proc format; ❷
value $numplus
'0004' = 'MEMORIAL HOSPITAL'
'0016' = 'CATHOLIC MED CTR (***CLOSED***)'
other  = [$num2nam35.] ❸
;
run;

title 'EXAMPLE 12 - FORMAT IN LIBRARY, FORMAT WITH MODIFICATIONS';
proc format; ❹
select $num2nam $numplus;
run;

data hospitals; ❺
input hospnum : $4. @@;
datalines;
0001 0004 0012 0015 0016
;
run;

title 'EXAMPLE 12 - REPORT WRITTEN USING A NESTED FORMAT';
proc report data=hospitals nowd; ❻
columns hospnum hospnum=hospnam;
define hospnam / 'name' format=$numplus.;
run;
```

EXAMPLE 12 - FORMAT IN LIBRARY, FORMAT WITH MODIFICATIONS ❼

FORMAT NAME: \$NUM2NAM LENGTH: 30 NUMBER OF VALUES: 7		
MIN LENGTH: 1	MAX LENGTH: 40	DEFAULT LENGTH 30 FUZZ: 0
START	END	LABEL (VER. V7 V8 22JUN2005:15:35:19)
0001	0001	ALBANY MEDICAL CENTER HOSPITAL
0002	0002	CHILDS HOSPITAL1
0005	0005	ST PETERS HOSPITAL
0012	0012	OSWEGO HOSPITAL
0016	0016	CATHOLIC MED CTR
0025	0025	COHOES MEMORIAL HOSPITAL
**OTHER**	**OTHER**	UNKNOWN

FORMAT NAME: \$NUMPLUS LENGTH: 35 NUMBER OF VALUES: 3		
MIN LENGTH: 1	MAX LENGTH: 40	DEFAULT LENGTH 35 FUZZ: 0
START	END	LABEL (VER. V7 V8 22JUN2005:15:35:19)
0004	0004	MEMORIAL HOSPITAL
0016	0016	CATHOLIC MED CTR (***CLOSED***)
**OTHER**	**OTHER**	[\$NUM2NAM35.]

EXAMPLE 12 - REPORT WRITTEN USING A NESTED FORMAT ❸

```
hosp
num  name
0001 ALBANY MEDICAL CENTER HOSPITAL
0004 MEMORIAL HOSPITAL
0012 OSWEGO HOSPITAL
0015 UNKNOWN
0016 CATHOLIC MED CTR (***CLOSED***)
```

Assume that the user-written format \$NUM2NAM is part of a format library ❶. Two changes must be made to the format prior to writing a report. A new format is created in the work library ❷. One addition is made (a new hospital is added, number 0004), and one change is made (hospital number 0016 is now closed). The labels for all other hospital numbers are to be taken from the old format ❸. Notice that a length is added to the format name. Without specification of a length, there would be a note in the LOG file...

NOTE: The \$NUM2NAM (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 \$NUM2NAM (in)format, or provide a sufficient DEFAULT= option.

To avoid this note, and also to avoid creating formats with truncated labels (if the length should be more than 40), specify a length for the nested format. PROC FORMAT is used to display the format in the format library (remember to pretend) and new format ❹. A data set with hospital numbers is created to use with the new format ❺ and the data set is printed using PROC REPORT ❻. When the second format, \$NUMPLUS, is displayed, only the modifications are shown. The OTHER condition lists the format in the library ❼. The output from PROC REPORT shows that both the old and new labels were used where appropriate ❸.

SAS-supplied formats can also be nested within another format. The next example shows a format that is intended to label a given range of days as 'OK', but display dates outside the range with a SAS date format.

```
* EXAMPLE 13 - NESTING FORMATS (SAS-SUPPLIED);
title 'EXAMPLE 13 - NESTED SAS-SUPPLIED FORMATS';
proc format;
value chk_date '01FEB1999'd - '31JUL1999'd = 'OK' other = [date9.]; ❶
select chk_date; ❷
run;

data mydates; ❸
format dt chk_date.;
input id : $3. dt : mmddyy. @@;
datalines;
123 01301999 234 02101999 456 07311999 789 12252000
;
run;

title 'EXAMPLE 13 - DATES OUTSIDE ACCEPTABLE RANGE ARE PRINTED'; ❹
proc report data=mydates nowd;
column id dt;
define id / 'ID #' width=4;
define dt / 'DATE' format=chk_date.;
run;
```

EXAMPLE 13 - NESTED SAS-SUPPLIED FORMAT ❺

FORMAT NAME: CHK_DATE LENGTH: 9 NUMBER OF VALUES: 2			
MIN LENGTH: 1		MAX LENGTH: 40	
		DEFAULT LENGTH 9 FUZZ: STD	
START	END	LABEL (VER. V7 V8 22JUN2005:15:53:01)	
14276	14456	OK	
**OTHER**	**OTHER**	[DATE9.]	

EXAMPLE 13 - DATES OUTSIDE ACCEPTABLE RANGE ARE PRINTED ❻

```
ID #    DATE
123    30JAN1999
234     OK
456     OK
789    25DEC2000
```

The numeric format CHK\_DATE is created with a range of acceptable dates and an OTHER condition that specifies the use of a SAS-supplied format (DATE9.) for dates not in the acceptable range ❶. A SELECT statement is used to display the format ❷. The data set MYDATES is created ❸, and PROC REPORT is used to display the data set using the recently created format ❹. The OTHER condition shows the SAS-supplied format ❺. (Where are the dates for the acceptable values? What are those numbers - 14276 and 14456?) The output from PROC REPORT shows two acceptable dates (OK) and the values of two dates out of acceptable range.

PRELOADED FORMATS

Three SAS PROCs allow one to preload formats: MEANS (or SUMMARY); TABULATE; REPORT. Preloading formats can be used to force a procedure to display all the ranges in a format even when a data set does not contain values that fall into one or more ranges. The next example shows how to use a preloaded format with PROC MEANS. The data set CLASS from the SASHELP library is used (provided with BASE SAS) and it contains information on gender, age, height, and weight for nineteen people.

```
* EXAMPLE 14 - PRELOADED FORMATS IN PROC MEANS;
proc format;
value age ❶
low - 15 = '<16'
16 - high = '16+'
;
run;

title 'EXAMPLE 14 - PROC MEANS WITHOUT A PRELOADED FORMAT';
proc means data=sashelp.class mean maxdec=1; ❷
var weight;
class sex age;
format age age.;
run;

title 'EXAMPLE 14 - PROC MEANS WITH A PRELOADED FORMAT';
proc means data=sashelp.class mean maxdec=1 completetypes;❸
var weight;
class sex age / preloadfmt; ❹
format age age.;
run;
```

EXAMPLE 14 - PROC MEANS WITHOUT A PRELOADED FORMAT ❺

Analysis Variable : Weight			
Sex	Age	N	Mean
		Obs	
F	<16	9	90.1
	16+	0	.
M	<16	9	104.4
	16+	1	150.0

EXAMPLE 14 - PROC MEANS WITH A PRELOADED FORMAT ❻

Analysis Variable : Weight			
Sex	Age	N	Mean
		Obs	
F	<16	9	90.1
	16+	0	.
M	<16	9	104.4
	16+	1	150.0

A format is created that will be used to put observations into groups based on age ❶. PROC MEANS is run using the variable AGE is a CLASS statement plus the age format created in the previous job step ❷. Then, PROC MEANS is run again using the COMPLETETYPES ❸ and PRELOADFMT options ❹. The output from the first PROC shows that there are no females (F) in the data set who are 16 years old or older ❺. Since there are no such observations, there is no mean height calculated for that group. However, when the format was preloaded using the PRELOADFMT option on the CLASS statement in combination with the PROC option COMPLETETYPES, all age levels are shown for both genders ❻. Without the COMPLETETYPES option, there would be no combination of F/16+ in the output plus a warning message in the SAS LOG...

**WARNING:** PreloadFmt will have no effect on the output without one of the following options: "completeTypes", "order=data", or the class statement option "exclusive".

You can see from the warning that there are other options (ORDER=DATA, EXCLUSIVE) that can be used in combination with PRELOADFMT. This example is just meant to make you aware that you can use a format to determine the content of output from three PROCs rather than just relying on the content of your data set(s). This is especially useful if you create reports that must look consistent from one report to the next. Preloaded formats can be used to control the appearance and assure that the look is consistent, regardless of the data used.

MULTILABEL FORMATS

If you try to create a format and ranges of values to be formatted overlap, you will get an error message in the SAS LOG and no format will be created. For example...

```
proc format;
value age
15-17 = '15-17'
18-19 = '18-19'
15-19 = '15-19'
;
run;
```

produces a message in the SAS LOG (*What exactly is FUZZ?*)...

```
ERROR: These two ranges overlap: 15-17 and 15-19 (fuzz=1E-12).
NOTE: The previous statement has been deleted.
```

You can create and use formats with overlapping ranges if you use some procedure options. The next example uses the CLASS data set from the SASHELP library (also used in the previous example that demonstrated preloaded formats).

```
* EXAMPLE 15 - MULTILABEL FORMAT;
* CREATE A FORMAT WITH OVERLAPPING RANGES - USED THE MULTILABEL OPTION;
proc format;
value age (multilabel) ❶
11 - 13 = '11-13' ❷
11      = '  11'
12      = '  12'
13      = '  13'
14 - 16 = '14-16'
14      = '  14'
15      = '  15'
16      = '  16'
low - high = 'TOTAL'
;
run;

title 'EXAMPLE 15 - PROC MEANS USING THE MULTILABEL FORMAT WITHOUT OPTIONS';
proc means data=sashelp.class mean maxdec=1; ❸
var weight;
class age;
format age age.;
run;

title 'EXAMPLE 15 - PROC MEANS USING THE MULTILABEL FORMAT WITH OPTION';
proc means data=sashelp.class mean maxdec=1;
var weight;
class age /mlf; ❹
format age age.;
run;
```

EXAMPLE 15 - PROC MEANS USING THE MULTILABEL FORMAT WITHOUT OPTIONS ❺

Analysis Variable : Weight

Age	N	Mean
Obs		
TOTAL	19	100.0

EXAMPLE 15 - PROC MEANS USING THE MULTILABEL FORMAT WITH OPTION ❻

Analysis Variable : Weight

Age	N	Mean
Obs		
11	2	67.8
12	5	94.4
13	3	88.7
14	4	101.9
15	4	117.4
16	1	150.0
11-13	10	87.4
14-16	9	114.1
TOTAL	19	100.0

In order to create a format with overlapping ranges, the MULTILABEL option is used ❶. The format is arranged in an order that you would like to see used in a procedure ❷. PROC MEANS is run using the multilabel format to group observations based on the value of the CLASS variable AGE ❸. The MLF option is added to the CLASS statement in PROC MEANS ❹. The output from PROC MEANS shows that it is not enough to simply create and use a multilabel format ❺. Once the MLF option is added to the CLASS statement, all the ranges in the format appear in the PROC MEANS output ❻.



Notice the order in the last output. It is formatted order, but not in the order that was set up in PROC FORMAT. The default behavior of PROC FORMAT is to sort the format in range order, not to leave the format in its original order. If you want to maintain the order shown in PROC FORMAT, you could add some text to the labels...

```
proc format;
value age (multilabel)
11 - 13 = '1: 11-13'
11      = '2:   11'
12      = '3:   12'
13      = '4:   13'
14 - 16 = '5: 14-16'
14      = '6:   14'
15      = '7:   15'
16      = '8:   16'
low - high = '9: TOTAL'
;
run;
```

Now the formatted order would result in output that looks exactly as the format appears. There is an easier way that also eliminates the need for the extra text in the format labels.

```
* EXAMPLE 16 - MULTILABEL FORMAT;
* CREATE A FORMAT WITH OVERLAPPING RANGES - USED THE MULTILABEL OPTION;
proc format;
value age (multilabel notsorted)①
11 - 13 = '11-13'
11      = '   11'
12      = '   12'
13      = '   13'
14 - 16 = '14-16'
14      = '   14'
15      = '   15'
16      = '   16'
low - high = 'TOTAL'
;
run;

title 'EXAMPLE 16 - PROC MEANS USING THE MULTILABEL FORMAT WITH OPTIONS';
proc means data=sashelp.class mean maxdec=1;
var weight;
class age /mlf preloadfmt order=data; ②
format age age.;
run;
```

EXAMPLE 16 - PROC MEANS USING THE MULTILABEL FORMAT WITH OPTIONS ③  
Analysis Variable : Weight

Age	N Obs	Mean
11-13	10	87.4
11	2	67.8
12	5	94.4
13	3	88.7
14-16	9	114.1
14	4	101.9
15	4	117.4
16	1	150.0
TOTAL	19	100.0

Adding the NOTSORTED option in PROC FORMAT leaves the format in the order as it appears ①. The MLF option is still needed on the CLASS statement in PROC MEANS, but the format is preloaded and the ORDER=DATA option is added ②. The result of using this combination of matches the order of the ranges in PROC FORMAT ③.

USER-WRITTEN INFORMATS

Though there are a number of SAS-supplied informats that permit you to read data a large variety of data types, there are occasions when user-written informats can help you avoid using data step statements when converting raw data into a SAS data set. For example, what if you have a data file of cholesterol reading that contains a mix of the following: values that are thought to be correct; values that are known to be either too low or too high given a set of rules as to an allowable data ranges;

the text N/A (not available) when values are missing. What you would like to do is create a data set where the 'correct' values are unchanged and all others are converted missing. You would also like to be able to differentiate among those values that are truly missing (coded as N/A) and those that are out of the acceptable range.

```
* EXAMPLE 17 - USER-WRITTEN INFORMAT TO 'CLEAN' DATA;
proc format;
invalue chol ❶
N/A = .
low - 49 = .L
50 - 499 = _same_
500 - high = .H
;
value chol ❷
. = 'MISSING'
.L = 'TOO LOW'
.H = 'TOO HIGH'
50 - 129 = 'OPTIMAL/NEAR OPTIMAL'
130 - 159 = 'BORDERLINE HIGH'
160 - 499 = 'HIGH/VERY HIGH'
;
run;

data test;
input id : $3. chol : chol. @@; ❸
datalines;
123 N/A 234 10 345 600 456 80 567 150 678 280
;
run;

title "EXAMPLE 17 - DATA 'CLEANED' WITH AN INFORMAT";
proc report data=test nowd; ❹
columns id chol chol=chol_label;
define chol / display;
define chol_label / display 'cholesterol group' format=chol.;
run;

title;
proc means data=test n mean maxdec=1; ❺
run;
```

EXAMPLE 17 - DATA 'CLEANED' WITH AN INFORMAT ❻

id	chol	cholesterol group
123	.	MISSING
234	L	TOO LOW
345	H	TOO HIGH
456	80	OPTIMAL/NEAR OPTIMAL
567	150	BORDERLINE HIGH
678	280	HIGH/VERY HIGH

```
Analysis Variable : chol ❼
N          Mean
-----
3          170.0
-----
```

An informat is created using PROC FORMAT ❶. The text 'N/A' is converted to missing, values outside the acceptable range are assigned special missing values, '.L' is too low and '.H' is too high. All other values are to remain the same and that is accomplished by using '\_SAME\_' as the informatted value. A format is also created to both label and group data values ❷. The data set TEST is created and the informat CHOL is used to clean the raw data as it is read ❸. PROC REPORT is used to display both the data value and its formatted value ❹. PROC MEANS computes the mean cholesterol ❺. The output from PROC REPORT shows how the raw data has been converted by the informat ❻, and the output from PROC MEANS shows that the first three values are all treated as missing data, not just the value that had been coded as 'N/A' ❼.

Without the user-written informat, the combination of text and numbers would force you to first read all the values as character data. You would then have to write a series of statements to convert the 'N/A' to missing, assign special missing values to readings outside the acceptable range, then finally convert the character data to numeric.

## BACK TO BASICS

There are a number of 'basics' that one should know when working with formats/informats. The following list is not exhaustive, is not in priority order, but a SAS user should be aware of all of the following.

### **Where are Formats/Informats Stored**

Formats and informats are stored in a catalog. SAS documentation defines a catalog as follows...*SAS catalogs are special SAS files that store many different kinds of information in smaller units called catalog entries. Each entry has an entry type that identifies its purpose to the SAS System.* The default name for the catalog that SAS uses to store formats/informats is FORMATS. Unless you specify otherwise, the catalog is created in the WORK library during any given SAS session. Just as you can create temporary or permanent SAS data sets, you can create temporary or permanent formats. If you want to create permanent formats, the most convenient method is to create them in a library name LIBRARY, for example...

```
libname library 'c:\sas\myformats';
proc format library=library;
value $gender
'1' = 'MALE'
'2' = 'FEMALE';
run;
```

either creates or adds to a catalog name FORMATS in the folder C:\SAS\MYFORMATS. The advantage of using the LIBNAME LIBRARY is that the default order that SAS uses to search for formats/informats is the library WORK, then the library named LIBRARY. SAS also looks for the formats/informats in a catalog named FORMATS is either of those libraries. You can create formats in other locations and in catalogs with names other than FORMATS. However, if you do so, you will have to use the FMTSEARCH option to tell SAS where your formats/informats are located.

### **What User-Written Formats/Informats are Available**

Since formats/informats are stored in a catalog, you can use PROC CATALOG to display the names of available formats. If you want a list from the current SAS session (temporary formats), you can use...

```
proc catalog c=work.formats;
contents;
run;
quit;
```

The format/informat names will be listed together with a type. The types are FORMAT, FORMATC, INFMT, and INFMTC (the 'C' suffix identifies a character format/informat). You can also ask to list only specific format/informat types by specifying an entry type as a PROC option...

```
proc catalog c=work.formats et=format;
contents;
run;
quit;
```

Since the CONTENTS statement does not support an entry type option, you can only list one entry type for each use of PROC CATALOG. You can use various PROC CATALOG options to manage the contents of format/informat catalogs.

### **What Rules Does a Format/Informat Contain**

You can use the SELECT statement in PROC FORMAT to view the contents of a particular format. For example, to view the contents of the format \$GENDER in the WORK library...

```
proc format;
select $gender;
run;
```

Notice that no period is used in the format name in the SELECT statement. To look at the contents of a format/informat in a library other than WORK, you must specify the library name. Also, to look at the contents of all the formats, you can use the FMTLIB option in place of the SELECT statement, for example...

```
libname library 'c:\sas\myformats';
proc format library=library fmtlib;
run;
```

will display the content of all the formats in a permanent format library.

***Does It Matter Where I Put a Format Statement in a Data Step***

If you assign a format to a variable in a data step using a FORMAT statement, the assignment is permanent in that the data set you are creating contains information that the format and variable are linked. The location of the FORMAT statement sometimes makes a difference.

```
*EXAMPLE 18 - FORMAT LOCATION DOES MATTER;
proc format;
value $answer ❶
"YES" = '1'   "NO" = '2' "DON'T KNOW" = '3'   other = '4'   " " = '0'
;
run;

data results;
format answer $answer.; ❷
input answer $15.; ❸
datalines;
YES
DON'T KNOW
NOT INTERESTED

NO
;
run;

title 'EXAMPLE 18 - CONTENTS OF DATA SET RESULTS';
proc contents data=results;
run;

title 'EXAMPLE 18 - DATA SET RESULTS';
proc print data=results;
run;

EXAMPLE 18 - CONTENTS OF DATA SET RESULTS
Alphabetic List of Variables and Attributes
#   Variable   Type    Len    Format
1   answer     Char     1     $ANSWER. ❹

EXAMPLE 18 - DATA SET RESULTS ❺
Obs    answer
1       4
2       4
3       4
4       0
5       4
```

A format is created to convert text responses to numbers ❶. The FORMAT statement in the data step is located prior to the INPUT statement ❷. The INPUT statement specifies a character variable with a length of 15 ❸. The results of PROC CONTENTS show that the length of the variable ANSWER is 1 (not 15) ❹. The results of PROC PRINT are not what might be expected ❺. Since the length of ANSWER is only 1, the real values of ANSWER for the five observations are: Y, D, N, ., N. The missing value is assigned a formatted value of '0', while the remaining values are assigned a value of '4' based on the OTHER condition in the format. So, a format is not just an instruction that SAS uses to write data values. In this example, it is also an instruction how to store values and led to unexpected results in the printing of the data set. In the next example, the location of the FORMAT statement remains the same, but the format and data are changed.

```
proc format;
value $answer
'1' = "YES"   '2' = "NO"   '3' = "DON'T KNOW"   '4' = "OTHER"
;
run;

data results;
format answer $answer.;
input answer : $1. @@;
datalines;
1 2 3 4
;
run;
```

In this case, you get the expected results from PROC PRINT, but the length of the variable ANSWER is 10 (not 1), the length of the longest format label. That is too many bytes to store 1 character.

## CONCLUSION

There are many uses of formats in SAS beyond the substituting labels for the values of variables or the grouping of observations based on the values of variables. Features such as CNTLIN data sets, PUTN and PUTC functions, and preloaded and multilabel formats offer a SAS user much flexibility in the creation and use of formats. Informats make it easy to screen raw data for unacceptable values and offer a convenient alternative for converting character data strings into values of numeric variables. The contents of formats libraries and of formats and informats are easily displayed using PROC CATALOG and PROC FORMAT.

## REFERENCES

Recent SUGI and NESUG proceedings contain a number of papers on the creation and use of formats/Informats. This is a selected list and all citations are linked to papers available on the web.

SUGI 30 (2005)

[\*PROC FORMAT ? Not Just Another Pretty Face\*](#)  
[\*PROC FORMAT: A Speedy Alternative to Sort/Merge\*](#)  
[\*My Friend the SAS Format\*](#)

SUGI 29 (2004)

[\*\(In\)Formats \(In\)Decently Exposed\*](#)  
[\*Building and Using User Defined Formats\*](#)

SUGI 28 (2003)

[\*The BEST. Message in the SAS® Log\*](#)  
[\*Describing and Retrieving Data with SAS® formats\*](#)

SUGI 27 (2002)

[\*More than Just Value: A Look Into the Depths of PROC FORMAT\*](#)  
[\*PROC FORMAT in Action\*](#)  
[\*Powerful Techniques For Data Processing Using Formats\*](#)

SUGI 26 (2001)

[\*Eight PROC FORMAT Gems\*](#)  
[\*SAS Formats: Making the Best of a Bad Situation\*](#)  
[\*Merging Small Data sets To Large Unsorted Tape Data sets without a Sort Step, Using SAS Formats\*](#)

NESUG 17 (2004)

[\*Using PROC FORMAT for data validation and clean-up\*](#)  
[\*The Power of PROC FORMAT; Updated for SAS8 and SAS9\*](#)

NESUG 14 (2001)

[\*INVALID: a Data Review Macro Using PROC FORMAT Option OTHER="INVALID" to Identify and List Outliers\*](#)  
[\*Validating And Updating Your Data Using SAS Formats\*](#)  
[\*Dynamically Build a PROC FORMAT for Subsetting Large Datasets\*](#)  
[\*What User Defined Formats Are Available?\*](#)

NESUG 13 (2000)

[\*The Power of PROC FORMAT\*](#)  
[\*Ten Things You Should Know About PROC FORMAT\*](#)  
[\*New Ways to Update Old Formats\*](#)  
[\*Create Custom Numeric Formats with the PROC FORMAT Picture Statement\*](#)  
[\*SAS Software Formats: Going Beneath the Surface\*](#)

## TRADEMARK CITATION

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are registered trademarks or trademarks of their respective companies.

## CONTACT INFORMATION

The author can be contacted using e-mail... [msz03@albany.edu](mailto:msz03@albany.edu)