

# Using Formats in SAS

Brian Kreeger  
Ingenix  
December 11, 2008

# What are Formats Used For?

- Formats allow us to map one value into another; we can easily label and group data
- Two general classes of Formats: informats and formats
- Informats determine how raw data is read and stored (informats convert)
- Formats determine how data is output (formats print)

# Proc Format

- The Format Procedure allows us to create our own formats: we can specify them, create them from a data set or use an create a dataset to create one for us

# Rules (SAS 9)

- Numeric Format names must be a valid SAS name, and can be 32 characters in length
- Character Format names also must be valid SAS names, must start with \$, and can have 31 characters
- User-defined format names can't be the same as existing SAS formats
- **Formats can't begin or end in numbers**

# Example: Grouping

- We have 23 states from across the USA split into 3 regions. How can we efficiently identify the regions for analysis?
- Old School: If-Then-Else logic
- New School: Proc Format

# Example: Grouping

- Old School

```
/* Region 1: WA, OR, CA, MT, ID, NV
Region 2: ND, SD, NE, KS, OK, TX, MN, IA, MO, AR
Region 3: FL, GA, AL, TN, PA, OH, IN */

if state in ('WA' 'OR' 'CA' 'MT' 'ID' 'NV') then region = 1;
else if state in ('ND' 'SD' 'NE' 'KS' 'OK' 'TX' 'MN' 'IA' 'MO' 'AR')
    then region = 2;
else if state in ('FL' 'GA' 'AL' 'TN' 'PA' 'OH' 'IN') then region = 3;
else region = 99;
```

# Example: Grouping

- New School

```
proc format;  
value $reg_state  
'WA', 'OR', 'CA', 'MT', 'ID', 'NV' = 1  
'ND', 'SD', 'NE', 'KS', 'OK', 'TX', 'MN', 'IA', 'MO', 'AR' = 2  
'FL', 'GA', 'AL', 'TN', 'PA', 'OH', 'IN' = 3  
other = 99; run;  
  
data temp2; set temp;  
region = put(state,$reg_state.);  
run;
```

# Example: Grouping (results)

25,000 records, 5 variables

Old School:

```
DATA statement used (Total process time):  
real time          0.06 seconds  
user cpu time      0.01 seconds  
system cpu time    0.00 seconds
```

New School:

```
DATA statement used (Total process time):  
real time          0.04 seconds  
user cpu time      0.01 seconds  
system cpu time    0.00 seconds
```

# Example: Data Cleaning

- Another quick example: checking validity of the data values of a field
- Checking credit scores, ranging from 250 to 800

```
proc format;  
value good_range  
250 - 800 = 'good'  
other = 'not good';  
run;  
proc freq data=temp;  
table cbr_score;  
format cbr_score good_range.;  
run;
```

# Example: Data Cleaning

- Extract records with scores that fell out of acceptable range:

```
data weird_scores; set temp;  
where put(cbur_score, good_range.) ne 'good';  
run;
```

# Proc Format Options

- FUZZ= Option : allows for ranges without having to specifically code for them

```
value test (fuzz=.1)
1 = 'A'      0.9 - 1.1
2 = 'B'      1.9 - 2.1
3 = 'C';     2.9 - 3.1
```

- Default Fuzz = 1e-12
- JUST Option: left justifies variable before comparison to range
- UPCASE Option: converts all characters to uppercase before comparison to range

# Multilabel Formats

- Procs Means, Summary and Tabulate allow for multilabel formats

```
Value sales_grp (multilabel)
0-2 = 'very low'
3-5 = 'moderate'
6-9 = 'high'
9 = 'very high';
```

- Need to use /mlf option on Class statement

# Formats from Datasets

- The CNTLIN= allows us to create a format without coding a Proc Format statement
- The CNTLIN is a regular SAS dataset, and each record contains the equivalent of a range-value pair in Proc Format
- In this example, we create a format linking a facility name and facility number

```
data control (keep=fmtname start label);  
set facility_data (keep=fac_number fac_name);  
retain fmtname '$facility';  
rename fac_number=start fac_name=label;  
run;  
  
proc sort data=control nodupkey;  
by start;  
run;  
  
proc format cntlin=control;  
run;
```

# Datasets from Formats

- To write out a format to a dataset, use **CNTLOUT=**

```
proc format  
  library=work  
  cntlout=regions (where=(fmtname='reg_state_name')) ;  
run;
```

# Picture Formats

- Picture Formats use a template of predefined characters to place the value
- Need to use PICTURE statement

```
proc format;  
  picture phone  
  2000000 - 999999 = '999-9999'  
  9999999 <- 9999999999 = '999 999-9999'  
  other = 'miscoded';  
run;
```

# More about Phone Numbers

- Let's put parentheses around area code:

```
proc format;  
  picture phone  
  2000000 - 9999999 = '999-9999'  
  9999999 <- 99999999999 = '999) 999-9999'  
                                     (prefix = '(' )  
  other = 'miscoded';  
run;
```

# More about Picture Formats

- 9 is a single digit placeholder (blanks = 0)
- Use Fill= option to fill with exotic symbols, such as \*

```
low - high = '0000';
```

```
low - high = '9999';
```

```
low - high = '0000' (fill='*');
```

```
123 => 123  
0123  
*123
```

# Even More about Picture Formats

- If we need to convert values (i.e. currency) we can use the `Mult=` option

```
picture pay_mill  
low - high = '00.0M'  
          (prefix='$' mult=.00001);
```

```
picture pay_usd (round)  
low-high = '000,000,000US'  
          (prefix='$' mult=1.61);
```

# Creating a Format Library

- Storing Formats in a library for repeated use is easy:

```
libname stor_fmt '/home/formats';
```

```
proc format library=stor_fmt;  
value .....
```

- This creates a permanent format that won't have to be re-created at the start of each new session

# Using a Format Library

- When you specify a Format, SAS looks in its own Format Library first, then in any Format catalogs in the Work Library, if any
- Use `fmtsearch=` option to instruct SAS to search any format libraries you create and want to use and the order to search in

```
options fmtsearch = (stor_fmt work);
```

# References

- Andrew Karp, “My Friend The SAS Format” SUGI 30
- Andrew Karp, “Getting Into the Picture (Format)”, SUGI 31
- Art Carpenter, “Building and Using User-Defined Formats”, SUGI 29
- Jonas Bilenas, “I Can Do That with PROC FORMAT”, Global Forum 2008