

AN x64 PROCESSOR IS SCREAMING ALONG AT BILLIONS OF CYCLES PER SECOND TO RUN THE XNU KERNEL, WHICH IS FRANTICALLY WORKING THROUGH ALL THE POSIX-SPECIFIED ABSTRACTION TO CREATE THE DARWIN SYSTEM UNDERLYING OS X, WHICH IN TURN IS STRAINING ITSELF TO RUN FIREFOX AND ITS GECKO RENDERER, WHICH CREATES A FLASH OBJECT WHICH RENDERS DOZENS OF VIDEO FRAMES EVERY SECOND

BECAUSE I WANTED TO SEE A CAT JUMP INTO A BOX AND FALL OVER.



I AM A GOD.



Using Formats in SAS

Brian Kreeger, Ingenix
Twin Cities SAS Users Group
December 17, 2009



What are Formats Used For?

- As with any programming exercise:
KNOW YOUR DATA!
- 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 or create them from a data set
- They can be stored permanently and recalled for later use



Existing SAS Formats

- A few selected:

dollarw.d (dollar sign and commas)

percentw.d

ssnw. (converts number to ssn)

w.D (w=width, d=number of decimal places)

zw.d (writes leading zeros)

\$charw. (character with leading blanks)

\$w. (character)



How well do you know your formats?

- 2345.678 -> dollar9.2
\$2,345.68
- 0.7556 - > percent8.2
75.56%
- 'abcde' -> \$char8. and \$8.
' abcde'
'abcde '



User defined formats

- User defined formats can be used to:
 - Convert numeric variables to character values
 - Convert character strings into numbers (INPUT)
 - Convert character strings into different character strings
- *Incoming_value = formatted_value*



Rules (SAS 9)

- Numeric Format names must be a valid SAS name, and can be 32 characters in length (In SAS 8 and previous, Format names could only be 8 characters – further SAS won't print an ERROR message if they are too long)
- Consider: TooLongAName, TooLongAValue
- Character Format names also must be valid SAS names, must start with \$, and can have 31 characters (7 in SAS 8)
- User-defined format names can't be the same as existing SAS formats
- **Formats can't begin or end in numbers**
- A format will accept either numeric or character inputs, but will always produce **characters** (the \$ denotes that it is expecting character input)
[informats: takes character input and produces either character or numeric output;
\$ denotes it will produce character output]



Default Quoting

- SAS will use the \$ to quote the appropriate ranges and/or values (even if you don't in your code!)

```
proc format;  
value $testa  
'1' = 'A'  
'2' = 'B'  
'3' = 'C';
```

```
value $testb /* SAS will quote by default */  
1 = 'A'  
2 = 'B'  
3 = 'C';  
run;
```



Avoiding a Trap

- What's wrong with this format?

```
proc format;
```

```
value score
```

```
0 - 30 = 'Low'
```

```
30 - 60 = 'Medium'
```

```
60-100 = 'HIGH';
```

```
run;
```



Range Delimiters

- Use < to help this:

```
proc format;  
  
value score  
0 - 30 = 'Low'  
30 <- 60 = 'Medium'  
60 <- 100 = 'HIGH';  
run;
```

- <- eliminates the lower end range value; similar for -< or < - <



Example: Grouping (Lookup)

- 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 */

Data temp2; set temp;
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';
Run;
```



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 (Using SQL)

- To use this format in Proc SQL:

```
proc sql noprint;  
create table temp2 as  
select a.*, a.state as region format $reg_state.  
from temp as a;  
quit;
```



Example: Converting to Numeric

- A quick example illustrating converting a character value to numeric:

```
ID = INPUT (COMPRESS (SSN, '-' ), 9.);
```



Another Example

- Problem: I have 20 regions, 1-20, and I want to label 3 specific regions of interest (1, 11, and 17) and split the remainder into 1-10 and 11-20.

```
value $region_break  
'1' , '11', '17' = 'Good'  
'1' - '10' = 'Range A'  
'11' - '20' = 'Range B' ;
```



Nesting Formats

- Using Nested Formats to solve our problem:

```
value $reg_break  
'1', '11', '17' = 'Good'  
other          = [$reg_sub.] ;
```

```
value $reg_sub  
'1' - '10' = 'Range A'  
'11' - '20' = 'Range B' ;
```



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**
- **Watch out for overlapping ranges!** SAS will NOT issue ERROR message when FUZZ option causes ranges to overlap – and it's not always predictable how SAS will assign values
- JUST Option: left justifies variable before comparison to range (invalue)
- UPCASE Option: converts all characters to uppercase before comparison to range (invalue)



Picture Formats

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

```
proc format;  
  picture phone  
  2000000 - 9999999 = '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
```



Changing Separators

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

1234.56 becomes **1.234,56**



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);
```



Formats from Datasets

- The CNTLIN= allows us to create a format without coding a detailed Proc Format statement
- The CNTLIN is a regular SAS dataset, and each record contains the equivalent of a range(**start**)-value(**label**) 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;
```



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, and can be shared with other users



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
- ```
libname stor_fmt 'c:\sas\mystuff\';
options fmtsearch = (stor_fmt work);
```



# Multilabel Formats

- Proc 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



# 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
- Pete Lund, “More Than Just Value: A Look Into The Depths of PROC FORMAT”, SUGI 26
  
- [www.lexjansen.com](http://www.lexjansen.com)