Paper 072-2010

Eliminating Redundant Custom Formats (or How to Really Take Advantage of PROC SQL, PROC CATALOG, and the DATA STEP)

Philip A. Wright, University of Michigan, Ann Arbor, MI

ABSTRACT

Custom formats are an invaluable asset to the SAS® programmer. Their functionality provides for much more than simply a mechanism for explicitly labeling values in a dataset. There can be, however, a major limitation—the DATA STEP can only accommodate 4,096 formats at a time. It is unlikely that a SAS programmer would generate this many formats in code, but this is not the only method that generates formats. PROC IMPORT and other third party data *conversion* programs may well generate a distinct custom format for every variable in a data set, and data sets with more than 4,096 variables are not uncommon. Oftentimes, however, these formats can be quite redundant—the same coding scheme was used for many similar variables. Eliminating redundant custom formats may well get you below the 4,096 limit. PROC SQL, PROC CATALOG, and the DATA STEP are the tools that can be used to eliminate the redundant formats. Eliminating redundant formats should not be of concern for only those exceeding the 4,096 limit, but also be of concern for any SAS user with custom formats—"Eliminating redundant formats is always best."—Rick Langston, SAS' PROC FORMAT developer, at the Michigan SAS User's Group One Day Conference, May 2010.

This version of the paper departs slightly from previous versions; It incorporates some suggestions from Larry Hoyle, as well as not utilizing formats-associated datasets until processing is nearly complete: rather than keeping variable names associated with formats, a 'crosswalk' dataset is used to replace a dataset's redundant FORMATS CATALOG associations with the non-redundant FORMATS CATALOG using PROC DATASETS.

INTRODUCTION

SAS formats are much more than value labels for variables in a dataset. Formats can be used to recode variable values, used in lieu of table lookups, and even used to customize code at run time. Their widest use, however, is to render values of dataset variables in a manner much more descriptive than the values themselves. The use of formats is usually not problematic for smaller datasets but can be problematic when datasets comprise 4,096 variables or more: Automated production routines and third party data conversion programs often generate a distinct format, usually named after the variable itself, for every variable in the dataset and, as there is a limit of 4,096 formats (hereafter termed *the limit*) specified when DATA STEP code and other *procedures* (such as PROC DATASETS) are compiled, exceeding the limit with larger datasets is comparatively easy.

It is also fairly easy to designate the use of more than the limit in a large dataset by non-automated methods; designating the use of more than the limited number of formats can be done with PROC DATASETS as long you do not designate more than the limit in each distinct procedure invocation. You will, however, quickly find out when you are using a dataset with more than the limit—specifying a dataset that exceeds the formats limit will generate the following error:

```
ERROR 81-59: Limit of 4096 formats or informats in use in a single step has been
exceeded.
```

This error (as with all errors) will stop the DATA STEP in its tracks. Oftentimes, however, some of these formats actually generate the same rendered strings as other formats and are, therefore, redundant. This is especially true when the formats were generated by an automated process or third party program. Eliminating redundant custom formats using a series of PROC SQL, PROC DATASETS, PROC CATALOG, and DATA STEP*s* has the potential to get below the 4,096 format limit.

GENERATING A DATASET COMPRISED OF CUSTOM FORMAT DETAILS

Any standard use of a dataset with more than a combination of 4,096 informats and/or formats will generate the error message. The formats catalog itself, however, is not restricted to the limit. We also want to be careful not to modify either the original dataset or formats catalog and will instead use copies. The original dataset and formats catalog should be saved in a permanent library before we work with copies in the WORK library.

1. Initialize the folder/directory containing the *COPIES* of the dataset and FORMATS catalogue as the permanent library 'USER':

```
libname USER 'D:\My Documents\My SAS Files' ;
```

2. Once you have initialized a library which contains the *copy* of the FORMATS catalog with the redundant formats, you are then ready to use PROC FORMAT with the cntlout option to generate a dataset comprised of custom format metadata. We also make sure it is sorted as needed:

```
proc format
   library = USER
   cntlout = _FORMAT_METADATA
;
quit ;

proc sort
   data = _FORMAT_METADATA
;
by
   fmtname type start end label
;
run ;
```

Format metadata comprises the following data:

	Variable	Type	Len	Label
1	FMTNAME	Char	32	Format name
2	START	Char	16	Starting value for format
3	END	Char	16	Ending value for format
4	LABEL	Char	8	Format value label
5	MIN	Num	3	Minimum length
6	MAX	Num	3	Maximum length
7	DEFAULT	Num	3	Default length
8	LENGTH	Num	3	Format length
9	FUZZ	Num	8	Fuzz value
10	PREFIX	Char	2	Prefix characters
11	MULT	Num	8	Multiplier
12	FILL	Char	1	Fill character
13	NOEDIT	Num	3	Is picture string noedit?
14	TYPE	Char	1	Type of format
15	SEXCL	Char	1	Start exclusion
16	EEXCL	Char	1	End exclusion
17	HLO	Char	11	Additional information
18	DECSEP	Char	1	Decimal separator
19	DIG3SEP	Char	1	Three-digit separator
20	DATATYPE	Char	8	Date/time/datetime?
21	LANGUAGE	Char	8	Language for date strings

3. The non-uniqueness of the values for these variables identifies a redundant format. Accordingly, we next generate a key for each format from these values using BY FMTNAME processing within a DATA STEP; we concatenate each key value to the previous key value by fmtname until we reach the last record for that fmtname. Once we have concatenated the last value for the fmtname, we output a fmtname record with a newly-generated custom_format_string.

```
* GENERATE CUSTOM FORMAT STRINGS FOR EACH CUSTOM FORMAT ;
options nosource ;
data
   _custom_format_strings (
      keep = fmtname type custom_format_string string_length
      where = (missing(custom_format_string) NE 1)
   )
;
```

```
set
   _FORMAT_METADATA (
   )
;
by
   fmtname
;
attrib
   CUSTOM_FORMAT_STRING
       length = $ &_MAX_STRING_LEN
       format = $CHAR1024.
       label = 'Custom Format String'
   STRING_LENGTH
       length =   8
       format = comma12.0
       label = 'String Length'
;
retain
   custom_format_string (' ')
   format_count (0)
;

* NOW PRESERVING LEADING BLANKS FOR PROPER SORTING OF
  CHARACTER-RENDERED NUMERIC VALUES ;
custom_format_string =
   catt(
      '|',
      custom_format_string, '|',
      start, '|',
      end, '|',
      strip(label), '|',
      put(length, 8.0), '|',
      put(noedit, 1.0), '|',
      type, '|',
      sexcl, '|',
      eexcl
   )
;

if (last.fmtname)
then do ;
   format_count ++ 1 ;
   string_length = length(trim(left(custom_format_string))) ;
   if (string_length GE %eval(&_MAX_STRING_LEN - 1)) then put
      'WARNING: POTENTIAL STRING LENGTH OVERUN: ' FORMAT_COUNT= ;
   output ;
   custom_format_string = ' ' ;
end ;

run ;
```

We make sure we have enough room for long strings by initializing the Program Data Vector with the *attrib* statement *prior* to the *set* statement.

Read and concatenate a delimited custom format metadata record value for each fmtname.

Generate and output a record once each metadata variable value for a fmtname has been concatenated.

4. We then use PROC SQL to quickly find the length of the longest custom_format_string and export it to a MACRO varaiable. The MACRO variable is subsequently used to optimize the length of the custom_format_string variable.

```
* EXPORT MAXIMUM STRING LENGTH TO MACRO VARIABLE ;
proc sql noprint ;
select
   strip(put(max(string_length),5.0))
into
   :_maximum_string_length
from
   _custom_format_strings
;
```

```
quit ;
%put _MAXIMUM_STRING_LENGTH: &_MAXIMUM_STRING_LENGTH ;

* OPTIMIZE LENGTH OF CUSTOM FORMAT STRING ;
data
   _CUSTOM_FORMAT_STRINGS  (
      label = '_CUSTOM_FORMAT_STRINGS'
   )
;
attrib
   fmtname                label = 'Format Name'   length = $ 32
   type                   label = 'Variable Type'   length = $ 1
   custom_format_string   label = 'Custom Format String'
      length = $    &_MAXIMUM_STRING_LENGTH
      format = %nrbquote($CHAR%trim(&_MAXIMUM_STRING_LENGTH).)
;
set
   _CUSTOM_FORMAT_STRINGS (
      drop = string_length
   )
;
run ;
```

5. custom_format_string cannot be used for indexing as the range in values can be too extensive. That leaves us with sorting by custom_format_string and generating an index for fmtname, which we will eventually use for some BY processing.

```
proc sort
   data = _custom_format_strings
   out = _custom_format_strings (
      index = (fmtname)
   )
;
by
   custom_format_string
   fmtname
   type
;
run ;
```

6. It is often the case that the first fmtname in a series of redundant formats is suitable for all the redundant formats. We will prepend the varnum for each format to the custom format records so that the format associated with the first variable in the dataset will be the set of records used to generate a unique format.

```
proc sql ;
create view
   _ordered_custom_format_strings
as select
   variables.varnum,
   strings.fmtname as fmtname,
   variables.type as type,
   strings.custom_format_string as custom_format_string
from
   _CUSTOM_FORMAT_STRINGS strings
left join
   _FORMATTED_VARIABLE_METADATA variables
on
   (cats(strings.fmtname,'.') EQ compress(variables.format,'$'))
   and (strings.type EQ upcase(substr(variables.type,1,1)))
```

```
  order by
     custom_format_string,
     varnum
  ;
  quit ;
```

7. We now have everything we need to generate a dataset comprised of unique format strings:

```
data
   _unique_format_strings
;
set
   _ordered_custom_format_strings
;
by
   custom_format_string
;
if (first.custom_format_string) then output ;
run ;
```

8. It is now another occasion to use PROC SQL. We have a dataset which includes both the fmtname of non-redundant formats and only the first fmtname of the redundant formats. We can use the fmtnames from this dataset to select a unique set of custom format metadata records from the set of records initially extracted from the formats catalog with PROC FORMAT:

```
proc sql ;
create table
   _unique_format_values_info
as select
   *
from
   _FORMAT_METADATA
where
   fmtname in (
      select
         fmtname
      from
         _UNIQUE_FORMAT_STRINGS
   )
order by
   fmtname,
   type,
   start,
   end
;
quit ;
```

9. The _unique_format_values_info records contain all the information we need to generate a FORMATS catalogue comprised of unique formats. We do, however, need to get rid of the formats that are still in the catalog:

```
proc catalog
   catalog = USER.FORMATS
   kill
;
quit;
```

This code is a bit misleading—it does not delete the catalog. Instead it merely deletes any *entries* in the catalog.

10. Once the FORMATS catalog is emptied, it is just as easy to re-populate it with the metadata information for unique format entries as it was to extract initially extract the metadata:

```
proc format
   library = USER
   cntlin = _UNIQUE_FORMAT_VALUES_INFO
;
quit ;
```

11. We now have a FORMATS catalog that does not contain any duplicated formats. But what about the dataset? We need to generate a 'crosswalk' dataset that maps the redundant format names to the list of unique format names:

```
proc sql ;
 create table
    _FORMAT_CROSSWALK
 as select distinct
    _custom_format_strings.fmtname as fmtname,
    _custom_format_strings.type as type,
    _unique_format_strings.fmtname as unique_fmtname
 from
    _custom_format_strings custom
 left join
    _unique_format_strings unique on
       (custom.custom_format_string EQ unique.custom_format_string)
 ;

 create table
    _NUMBERED_FORMAT_PAIRS
 as select
    variables.varnum,
    variables.name,
    variables.type,
    crosswalk.unique_fmtname
 from
    _FORMATTED_VARIABLE_METADATA variables
 left join
    _FORMAT_CROSSWALK crosswalk
 on
    (compress(variables.format,'$.') EQ crosswalk.fmtname)
    and (upcase(substr(variables.type,1,1)) EQ crosswalk.type)
 order by
    varnum
 ;
 quit ;
```

12. The DESCRIPTOR portion of the dataset also needs updating so that it uses only the unique formats. It is at this point where we use PROC SQL, SAS' dictionary tables, and the crosswalk dataset to get around the '**ERROR 81-59: Limit of 4096**' Error. SAS' dictionary tables have everything we need to generate new FORMAT statements, and the crosswalk dataset has the specifications for the variable names, the redundant format names, and the unique format names. The first step to do this is to export the format statements to an indexed array of macro variables:

```
data _NULL_ ;
set
   _NUMBERED_FORMAT_PAIRS
   end = end_of_dataset
;
call symput('_varname_' || strip(put(_N_,6.0)), strip(name)) ;
```

```
if type EQ 'char' then
   call symput('_fmtname_' || strip(put(_N_,6.0)),
      strip(cats('$',unique_fmtname,'.'))
   )
;
else call symput('_fmtname_' || strip(put(_N_,6.0)),
      strip(cats(unique_fmtname,'.'))
   )
;
if end_of_dataset then call symput('_pairs_n', strip(put(_N_,6.0))) ;
run ;
```

13. We finally are able to use PROC DATASETS to issue the format specifications that are currently stored as a series of indexed macro variables:

```
proc datasets
   library = USER
   nolist
;
%put NOTE: CLEARING OLD VARIABLE--FORMAT PAIRINGS. ;
modify
   memname
;
format _ALL_ ;
run ;

%put NOTE: SPECIFYING NEW VARIABLE--FORMAT PAIRINGS. ;
modify
   memname ;
;
format
%do _i = 1 %to &_PAIRS_N ;
   &&_VARNAME_&_I &&_FMTNAME_&_I
%end ;
;
run ;
quit ;
```

This macro code merely loops through each indexed format assignment statement.

CONCLUSION

Even though there is a limit of 4,096 formats and it can be easy to exceed this limit with datasets comprising more than this number of variables, it is also possible to eliminate redundant custom formats and re-associate the variables with a collapsed set of custom formats. As with most things SAS, the preceding method is not necessarily the only method of eliminating redundant custom formats. This method does, however, highlight the use of PROC SQL to gain access to the metadata of datasets whose use of more than the formats limit would generate an error when used with the DATA STEP and other SAS *procedures.* In addition, standard SQL routines can be used to identify and eliminate redundant formats when based on keys generated from select format metadata variables generated by PROC FORMAT. The DATA STEP *and* PROC DATASETS, when used with the smaller metadata datasets, generate the intermediate datasets utilized by PROC SQL. Both PROC CATALOG and PROC DATASETS are used to manage the processing.

There are a couple of steps the author would like to add should he ever finds the time: The generation of a recursive macro that will determine the least number of format metadata fields required to generate unique keys; the generation of a macro that will generate versions of both the pre- and post-processed datasets (or sub-sampled datasets) comprised of only formatted values for subsequent comparison; and the generation of a macro that will recast labels to appropriate upper-lower case strings based on standard labeling conventions.

REFERENCES

- Bilenas, Jonas V (2008), "I Can Do That With PROC FORMAT," *Proceedings of SAS Global Forum 2008.* http://www2.sas.com/proceedings/forum2008/174-2008.pdf
- Carpenter, Arthur L. (2004), "Building and Using User Defined Formats," *Proceedings of the 29th annual SAS Users Group Conference.* http://www2.sas.com/proceedings/sugi29/236-29.pdf

- Karp, Andrew H. (2005) "My Friend the SAS Format," *Proceedings of the 30th annual SAS Users Group Conference.* http://www2.sas.com/proceedings/sugi30/253-30.pdf
- Lund, Pete (2001), "More than Just Value: A Look into the Depths of PROC FORMAT," *Proceedings of the 26th SAS Users Group Conference.* http://www2.sas.com/proceedings/sugi26/p018-26.pdf
- Patton, Nancy K. (1998) "In & Out of CNTL with PROC FORMAT," *Proceedings of the 23rd annual SAS Users Group Conference.* http://www2.sas.com/proceedings/sugi23/Coders/p68.pdf
- Shoemaker, Jack (2001) "Eight PROC FORMAT Gems," *Proceedings of the 26th annual SAS Users Group Conference.* http://www2.sas.com/proceedings/sugi26/p062-26.pdf

ACKNOWLEDGMENTS

I would like to thank Rick Langston for his encouragement of my production of this paper, the staff of ICPSR for their support, and the participants in the Michigan SAS Users group for their encouragement and suggestions.

ABOUT THE AUTHOR

Phil Wright graduated from the University of Michigan in 1986 with a Bachelors degree in Psychology. He first sat down in front of a PC when his first research project purchased their first PC and asked him to learn their word processing application (FinalWord) and then teach it to the rest of the staff. He has been in front of a PC ever since. Phil has been using SAS for over 15 years; specializing in the conversion of legacy data files, data management, reporting, PROC SQL, ODS, and Macro programming.

CONTACT INFORMATION

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

Name:	Philip A. Wright
Enterprise:	Inter-university Consortium for Political and Social Research (ICPSR), The Institute for Social Research (ISR), University of Michigan
Address:	P.O. Box 1248
City, State ZIP:	Ann Arbor, Michigan 48106-1248
Work Phone:	734-615-7886
Fax:	734-647-8200
E-mail:	pawright@umich.edu
Web:	http://www.icpsr.umich.edu