Paper 010-2010

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

Philip A. Wright, The University of Michigan

ABSTRACT

Custom formats are invaluable to the SAS® programmer. Their functionality provides for much more than simply a mechanism for explicitly labeling values in a data set. There can be, however, a major limitation—the DATA step can accommodate only 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 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.

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. Whereas setting `'options nofmterr;'` will get you around not having the designated formats loaded in a formats catalog in the current *fmtsearch* path, it will not get you around this error. One thing you *can* do is clear the format designations from the descriptor portion of the dataset using proc datasets:

```
proc datasets library = libname nolist ; modify memname ; format _ALL_ ; quit ;
```

The procedure, however, does not address the real problem—simply too many different formats specified for use with the variables in the dataset. 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 steps* has the potential to get below the format limit.

GENERATING A DATASET COMPRISED OF CUSTOM FORMAT DETAILS

Using a dataset with more than the limit of formats—any standard use of the dataset 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.

Appendix C is a processing flow diagram of the following steps. Readers may want to make a hard copy of the diagram as an aide in understanding the steps and tables of the processing described below.

1. Initialize the folder/directory containing the dataset and formats catalog as the permanent library ‘USER’:

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

   In general, I will use *libname* and *memname* to designate each of the two-part dataset names.

2. Use *proc SQL* and the dictionary tables to generate datasets comprised of metadata and avoid the '**ERROR 81-59: Limit of...**' message generated by the *data step* and other SAS *procedures*:

```
proc sql ;

* GENERATE DATASET COMPRISED OF FORMATS METADATA ;
create table
   work._variable_formats (index=(varname /unique))
as select
   name as varname                  label = 'Variable Name'    format = $32.,
   compress(format,'.') as format   label = 'Variable Format'  format = $49.
from
   dictionary.columns
where
   (libname EQ 'USER')
   and (memtype EQ 'DATA')
   and (memname EQ 'MEMNAME')
   and (format is not missing)
;

* GENERATE DATASET COMPRISED OF DATASET VARIABLE METADATA ;
create table
   work._contents
as select
   columns.name label = 'Variable Name',
   columns.type label = 'Variable Type',
   columns.length label = 'Variable Length',
   columns.varnum label = 'Variable Number',
   columns.label label = 'Variable Label',
   upcase(formats.format) as format label =
'Variable Format' length=49
from
   dictionary.columns
   left join work._variable_formats formats on
      (columns.name EQ formats.varname )
where
   (libname EQ 'USER')
   and (memtype EQ 'DATA')
   and (memname EQ 'MEMNAME')
order by
   varnum
;

quit ;
```

Specifying the formats metadata dataset as the left table in a left join limits the metadata to only the variables associated with the custom formats.

3. We need to make sure we will be working with the formats we need to work with, but we also need to make sure we do not change anything in the original formats catalog:

```
* COPY FORMATS CATALOG FROM PERMANENT LIBRARY TO WORK LIBRARY ;
options nonotes ;
proc catalog
   catalog = user.formats
;
copy out = work.formats ;
quit ;
options notes ;
```

(I turn off note logging temporarily so I do not get a list of all the formats that get copied. You may wish to do otherwise)

4. One of the greatest assets of *proc format* is that it is able to generate a data set comprised of the detailed information SAS uses to render detailed versions of the dataset values:

```
* GENERATE DATASET FROM FORMATS CATALOG ;
proc format
   library = work
   cntlout = work._formats_info ( index = (fmtname) replace = yes )
;
quit ;
```

   Appendix A contains a listing of _formats_info contents.

5. We then proceed to generate a dataset comprised of select metadata for only those dataset variables that utilize the custom formats by again using the fmtname from the _formats_info dataset we just generated as the left member of a *left join*:

```
* GENERATE DATASET COMPRISED OF VARIABLES WITH CUSTOM FORMATS ;
proc sql ;
create table
   work._custom_formatted_variables
as select distinct
   contents.varnum,
   contents.name as varname,
   formats.fmtname,
   formats.length
from
   work._formats_info formats
left join work._contents contents on
   (formats.fmtname EQ contents.format)
where
   (varname is not null)
   and (fmtname is not null)
order by
   varnum
;
quit ;
```

6. Now that we have our three primary datasets generated we are ready to begin generating unique keys. As our goal is to eliminate duplicated formats, the keys will comprise the key elements of our format records. These key elements may well vary depending upon the complexity of your formats. In general, however, the values for the *start, end,* and *label* fields should be enough to generate a key string for each format. The key strings are generated by simply appending the values of these fields first to one-another, and then concatenating all the strings for each format:

```
* GENERATE DATASET OF CUSTOM FORMAT KEYS ;
data
   work._custom_format_keys (
      keep = fmtname custom_format_key string_length
      where = (not missing(custom_format_key))
   )
;
set
   work._formats_info
;
by
   fmtname
;
attrib
   custom_format_key   length = $ &MAX_STRING_LEN   format = $CHAR1024.
   string_length          length =   8                format = COMMA12.0
;
retain
   custom_format_key (' ')
   string_count (0)
;
```

continuing the generation of unique format keys:

```
custom_format_key =
   strip(custom_format_key) || '^'
   || strip(start) || '^'
   || strip(end) || '^'
   || strip(label)
;
if (last.fmtname)
then do ;
   string_count ++ 1 ;
   string_length = length(trim(left(custom_format_key))) ;
   if (string_length ge %eval(&max_string_len - 1))
      then put 'warning: potential string length overun: ' string_count= ;
   output ;
   custom_format_key = ' ' ;
end ;
```

This code currently maintains the case of the value labels in the custom_format_key strings. Users might find more redundant formats should the labels be cast entirely to upper or lower case so as to eliminate any differences in capitalization from what might otherwise be identical labels. The wrapping of the current `strip()` functions and their arguments with either the `upcase()` or `lowcase()` function would accomplish this.

7. Memory limitations of subsequent joins utilizing custom_format_key necessitates the ordering of the dataset due the default length of the variable; subsequent joins utilizing fmtname then necessitate the use of an index for fmtname:

```
proc sort
   data = work._custom_format_keys
   out = work._custom_format_keys (
      index = (fmtname)
   )
;
by
   custom_format_key
;
run ;
```

8. Once we have a dataset comprised of custom format keys we can merge the keys back with the original _formats_info dataset:

```
* AUGMENT FORMATS INFO WITH CUSTOM FORMAT KEYS ;
data
   work._formats_info
;
merge
   work._formats_info
   work._custom_format_keys (
      keep = fmtname custom_format_key
   )
;
by
   fmtname
;
run ;
```

9. Now that each key is paired with an original custom format we can determine the frequency of each key and, subsequently, how many times the metadata for each format has been duplicated:

```
* GENERATE CUSTOM FORMAT KEY FREQUENCIES DATASET ;
proc freq
   data = work._custom_format_keys
   noprint
;
```

continuing the generation of the custom format string frequencies dataset:

```
tables
   custom_format_key
   / nocol nocum nopercent norow out = work._format_string_frequencies (
      drop = percent
   )
;
proc sort
   data = work._format_string_frequencies
;
by
   custom_format_key
;
run ;
```

10. We can now begin to bring together the metadata we need for both generating unique formats and link the unique formats to the appropriate variables in the dataset:

```
* AUGMENT CUSTOM FORMAT KEYS WITH VARIABLE INFO ;
proc sql ;
alter table
   work._custom_format_keys
add
   varname   char(32)   label='Variable Name'     format=$32.,
   varnum    num(8)     label='Variable Number'
;
update
   work._custom_format_keys keys
set
   varname = (
      select distinct name
      from work._contents contents
      where contents.format EQ keys.fmtname
   ),
   varnum = (
      select distinct varnum
      from work._contents contents
      where contents.format EQ keys.fmtname
   )
;
run ;
```

11. Once we have joined the variable name and variable numbers to each appropriate format key we can generate a dataset comprised of unique format keys:

```
* GENERATE DATASET OF UNIQUE FORMAT RECORDS ;
proc sort
   data=work._custom_format_keys
   force
;
by
   custom_format_key
   varname
;
data
   work._unique_format_records
;
set
   work._custom_format_keys
;
by
   custom_format_key
;
if (first.custom_format_key) then output ;
run ;
```

12. Now that we have a dataset comprised of records with unique format keys we can join it with the information required to generate formats:

```
* GENERATE DATASET COMPRISED OF UNIQUE FORMATS INFO ;
proc sql ;
create table
   work._unique_formats_info
as select
   unique.varnum,
   unique.fmtname,
   formats.start,
   formats.end,
   formats.label,
   formats.type,
   formats.eexcl,
   formats.sexcl
from
   work._unique_format_records unique
   left join work._formats_info formats on (
      unique.fmtname EQ formats.fmtname
   )
order by
   varnum,
   fmtname,
   start,
   end
;
quit ;
```

13. Remembering that we only want to change the temporary formats catalog in the WORK library and that we do not want to duplicate any formats, we delete the current entries:

```
* GENERATE FORMATS CATALOG COMPRISED OF UNIQUE FORMATS ;
options nonotes ;
proc catalog
   catalog = work.formats
   kill
;
quit ;
options notes ;
```

14. We should now be ready to populate the temporary formats catalog with unique formats. As well as generating a dataset comprised of format metadata, *proc format* is also able to generate formats from a dataset *comprised* of format metadata--his time using the `cntlin=` option:

```
proc format
   library = work
   cntlin = work._unique_formats_info (drop = varnum) ;
;
quit ;
```

You can also generate *proc format* statements using the _unique_formats_info dataset should you need or prefer to go that route. Appendix B contains *data step* code that will generate the statements.

15. Almost there! We now have unique formats in our formats catalog, but the dataset metadata is still referencing the previous set of duplicated formats. We again use *proc SQL* to join the information we need for variable-format pairs with the _custom_format_keys dataset:

```
* GENERATE DATASET COMPRISED OF NEW VARIABLE-FORMAT DESIGNATIONS ;
options
   ibufsize=32767
;   * optimizes processing ;
proc sql ;
alter table
   work._custom_format_keys
add
   new_format    char(49)    label = 'New Format'      format=$49.,
   type          char(4)     label = 'Variable Type'   format=$4.
;
```

continuing generation of new variable-format designations:

```
update
   work._custom_format_keys keys
set
   new_format = (
      select fmtname
      from work._unique_format_records records
      where records.custom_format_key EQ keys.custom_format_key
   ),
   type = (
      select type
      from work._contents contents
      where contents.name EQ keys .varname
   )
;
proc sort
   data = work._custom_format_keys
;
by
   new_format
   varnum
;
run ;

proc sql ;
create table
   work._new_variable_format_pairs
as select distinct
   varnum,
   varname,
   type,
   new_format
from
   work._custom_format_keys
order by
   varnum
;
quit ;
```

16. We finally use some macro code to export strings comprised of the new variable-format pairs to indexed macro variables, invoke *proc datasets*, and re-specify the variable formats specified in our copied, temporary dataset:

```
%macro reset_format_specifications() ;

   %local _i ;

   * EXPORT VARIABLE-FORMAT PAIRS TO INDEXED MACRO VARIABLES ;
   data
      _null_
   ;
   set
      work._new_variable_format_pairs
      end = last_observation
   ;
   if (type EQ 'num') then new_format = strip(new_format) || '.' ;
   else new_format = '$' || strip(new_format) || '.' ;

   call symput('_varname_' || strip(put(_N_,4.0)), strip(varname)) ;
   call symput('_format_' || strip(put(_N_,4.0)), strip(new_format)) ;
   if (last_observation) then call symput('_pairs_n', strip(put(_N_,4.0))) ;
   run ;
```

continuing preparing for the generation of dataset with new variable-format designations:

```
   proc copy
      in = user
      out = work
   ;
   select
      memname
   ;
   run ;

* GENERATE DATASET WITH NEW VARIABLE-FORMAT DESIGNATIONS ;
   proc datasets
      library = work
      nolist
   ;
   modify
      memname
   ;
   format
   %do _i = 1 %to &_PAIRS_N ;
      &&_VARNAME_&_I &&_FORMAT_&_I
   %end ;
   ;
   run ;
   quit ;

%mend reset_format_specifications ;

%reset_format_specifications() ;
```

This macro code issues distinct variable-format pairs to the SAS Processor

Macro coding is outside the scope of this paper but a plethora of examples and papers can be downloaded from SAS' support web site: http://support.sas.com

Technically, we are now done eliminating redundant formats and resetting the formats specified for the variables in the dataset. Practically, however, we really should check our work.

1. We again use *proc SQL* to both avoid the error message and extract the names and formats from both the original dataset in the USER library and the modified dataset in the WORK library:

```
* GENRATE VARIABLE-FORMAT PAIR DATASETS FOR SUBSEQUENT COMPARISON WITH PROC
SQL ;
proc sql ;

* GENERATE VARIABLE-FORMAT PAIR DATASET FROM USER LIBRARY DATASET ;
create table
   work._user_ds_contents
as select
   name,
   format
from
   dictionary.columns
where
   (libname = 'USER')
   and (memtype = 'DATA')
   and (memname = 'MEMNAME')
order by
   name
;
quit ;
```

continuing generation of select dataset metadata for subsequent comparison:

```
* GENERATE VARIABLE-FORMAT PAIR DATASET FROM WORK LIBRARY DATASET ;
proc sql ;
create table
   work._work_ds_contents
as select
   name,
   format
from
   dictionary.columns
where
   (libname = 'WORK')
   and (memtype = 'DATA')
   and (memname = 'MEMNAME')
order by
   name
;
quit ;
```

2. Finally, we make a comparatively quick and painless call to *proc compare*:

```
* COMPARE VARIBLE-FORMAT PAIR DATASETS ;
options pageno = 1 ;
proc compare
   base = work._user_ds_contents
   comp = work._work_ds_contents
   maxprint = 32767
   note
;
id name ;
run ;
```

PROC COMPARE listing output:

```
                        The COMPARE Procedure
    Comparison of WORK._USER_DS_CONTENTS with WORK._WORK_DS_CONTENTS
                             (Method=EXACT)

                          Data Set Summary

Dataset                          Created          Modified  NVar    NObs

WORK._USER_DS_CONTENTS  12AUG09:08:58:08  12AUG09:08:58:08     2    4661
WORK._WORK_DS_CONTENTS  12AUG09:08:58:08  12AUG09:08:58:08     2    4661


                        Variables Summary

              Number of Variables in Common: 2.
              Number of ID Variables: 1.


                       Observation Summary

          Observation      Base  Compare  ID

          First Obs           1        1  name=ABORT12
          First Unequal       1        1  name=ABORT12
          Last  Unequal    4661     4661  name=Y_CHAN11
          Last  Obs        4661     4661  name=Y_CHAN11
```

The datasets have only two variables and one of them *is* used as an ID variable!

Everything looks good—nice and equal

PROC COMPARE listing output (continued):

```
Number of Observations in Common: 4661.
Total Number of Observations Read from WORK._USER_DS_CONTENTS: 4661.
Total Number of Observations Read from WORK._WORK_DS_CONTENTS: 4661.

Number of Observations with Some Compared Variables Unequal: 4310.
Number of Observations with All Compared Variables Equal: 351.
```

again; nice and equal

reasonable counts

```
                    Values Comparison Summary

  Number of Variables Compared with All Observations Equal: 0.
  Number of Variables Compared with Some Observations Unequal: 1.
  Total Number of Values which Compare Unequal: 4310.


            All Variables Compared have Unequal Values

        Variable  Type  Len   Label          Ndif   MaxDif

        format    CHAR   49   Column Format  4310
                    Value Comparison Results for Variables

_____________________________________________________________________
                       ||  Column Format
                       ||  Base Value           Compare Value
  name                 ||  format                format
  ____________________ ||  ____________________+ ____________________+
                       ||
  ADADREMQ             ||  ADADREMQ.            ABORT12F.
  ADPTOTOO             ||  ADPTOTOF.            ADPTOTOF.
  ADPTOTO1             ||  ADPTOT1F.            ADPTOTOF.
  ADPTOTO2             ||  ADPTOT2F.            ADPTOTOF.
  ADPTOTO3             ||  ADPTOT3F.            ADPTOTOF.
.
.
.
```

just as they should be

names of redundant formats

names of distinct formats

If everything checks out, you should now be ready to finish by copying the revised dataset and formats catalog from your WORK library to a library dedicated to the permanent storage of the revised SAS files:

```
libname REVISED 'D:\My Documents\My SAS Files\Revised' ;

proc copy
   in = work
   out = revised
;
select
   memname
   formats
;
run ;
```

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. I would also like to thank Larry Hoyle for his feedback and suggestions.

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

APPENDIX A:

Listing of Formats Dataset Metadata

Variables in Creation Order				
#	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	64	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

APPENDIX B:

Data step code that generates *proc format* statements

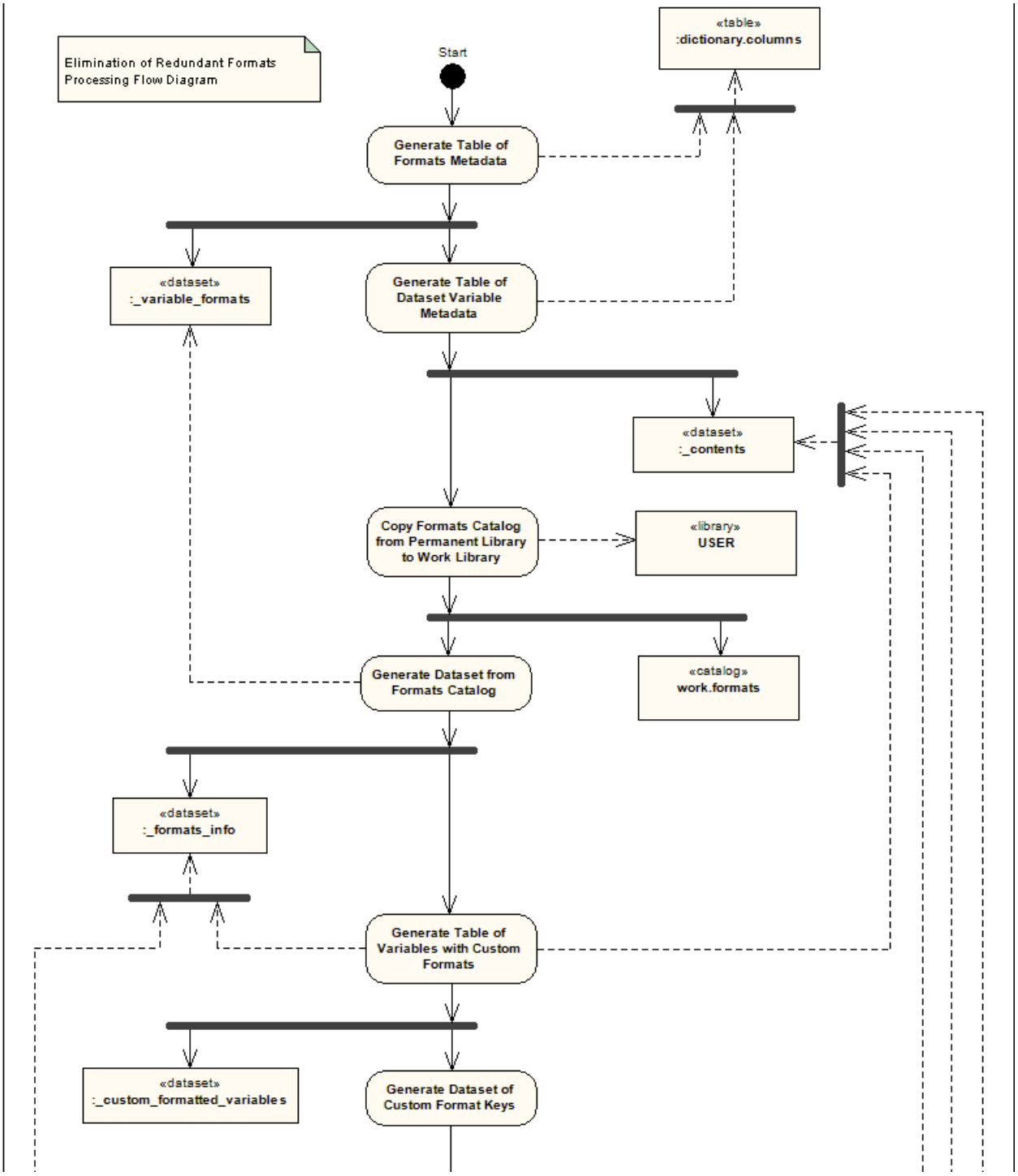
```
* GENERATE NEW PROC FORMAT STATEMENTS ;
data
   _NULL_
;
set
   work._unique_formats_info
   end = last_observation
;
by
   varnum
   fmtname
;
file
   'new_proc_format_statements.sas'
;
if (_N_ EQ 1) then put @1 'PROC FORMAT ;' ;
fmtname_len = length(strip(fmtname)) ;
start = strip(start) ;
start_len = length(strip(start)) ;
end = strip(end) ;
end_len = length(strip(end)) ;
label = tranwrd(tranwrd(label, "'", "''"), "''''", "''") ;
label_len = length(strip(label)) ;
select(type) ;
   when('N') do ;
      if (first.fmtname) then put @4 'value ' fmtname $VARYING. fmtname_len ;
      if (start EQ end) then put @7 start $VARYING. start_len " = '" label
$VARYING. label_len "'" ;
      else put @7 start $VARYING. start_len '-' end $VARYING. end_len " = '" label
$VARYING. label_len "'" ;
      if (last.fmtname) then put @4 ';' ;
   end ;
   when('C') do ;
      if (first.fmtname) then put @4 'value $ ' fmtname $VARYING. fmtname_len ;
      if (start EQ end) then put @7 "'" start $VARYING. start_len "' = '" label
$VARYING. label_len "'" ;
      else put @7 "'" start $VARYING. start_len "'-'" end $VARYING. end_len "' = '"
label $VARYING. label_len "'" ;
      if (last.fmtname) then put @4 ';' ;
   end ;
   otherwise error 'ERROR: NO FORMAT RECORD GENERATED' _N_= ;
end ;   * select(type) ;
if (last_observation) then put @1 'QUIT ;' ;
run ;

* GENERATE NEW FORMAT ASSIGNMENT STATEMENTS ;
data
   _NULL_
;
set
   work._custom_format_keys
   end = last_observation
;
by
   new_format
;
file
   'new_format_assignments.sas'
;
if (_N_ EQ 1) then put @1 'FORMAT' ;
```

Data step code that generates *proc format* statements (appendix B continued)

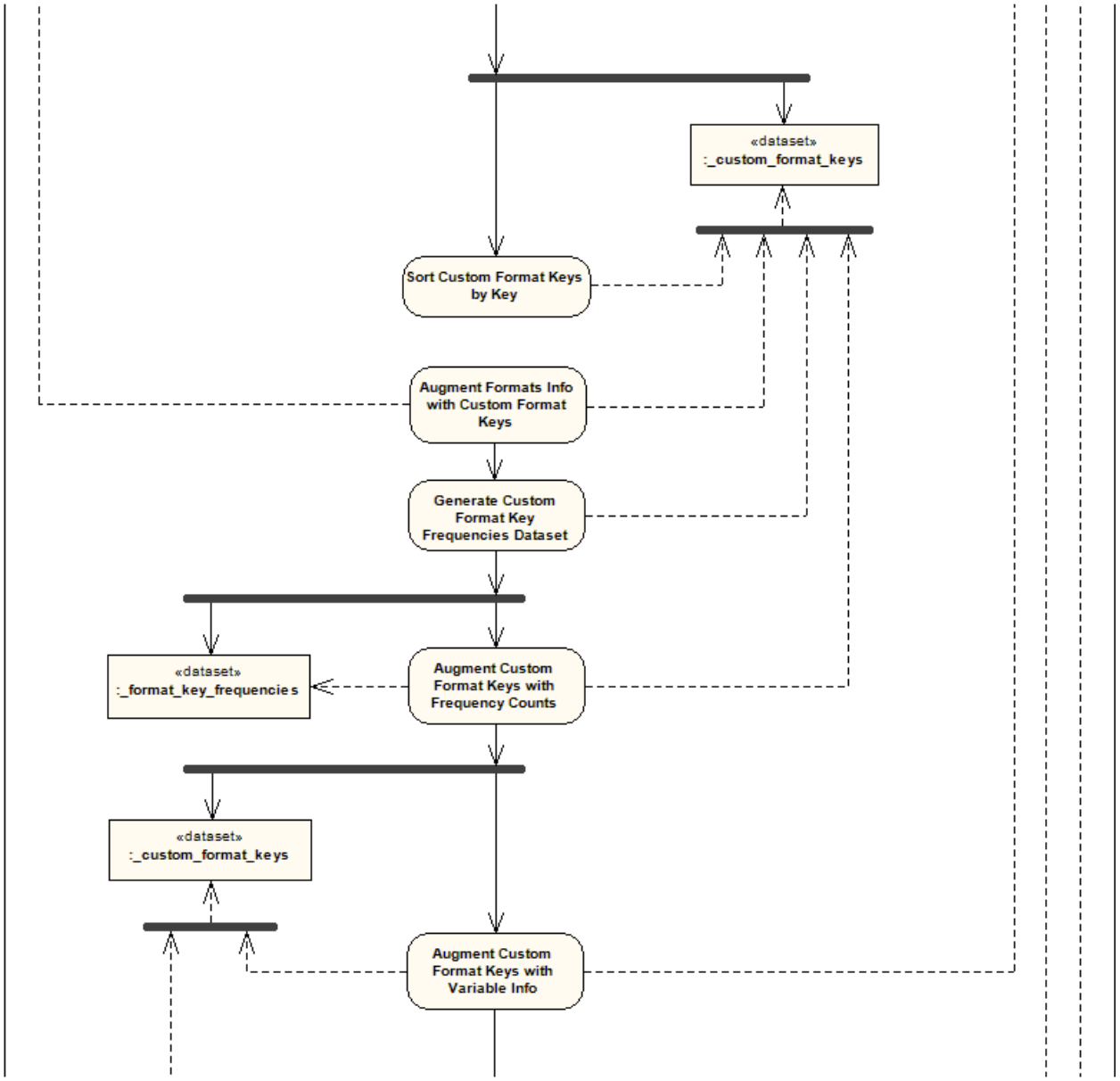
```
varname_len = length(strip(varname)) ;
put @4 varname $VARYING. varname_len ;
if (last.new_format)
then do ;
   new_format_len = length(strip(new_format)) ;
   select(type) ;
      when ('num') put @7 new_format $VARYING. new_format_len '.' ;
      when ('char') put @7 '$' new_format $VARYING. new_format_len '.' ;
      otherwise error 'ERROR: NO FORMAT ASSIGNED' _N_= ;
   end ;   * select(type) ;

end ;
if (last_observation) then put @1 ';' ;
run ;
```

APPENDIX C:

Elimination of Redundant Formats Processing Flow Diagram (page 1 of 3)



Elimination of Redundant Formats Processing Flow Diagram (continued – page 2 of 3)



Elimination of Redundant Formats Processing Flow Diagram (continued – page 3 of 3)