Paper 3301-2016

Masking data to obscure confidential values: a simple approach

Bruce Gilsen, Federal Reserve Board, Washington, DC

ABSTRACT

When I help users design or debug their SAS ® programs, they are sometimes unable to provide relevant SAS data sets because they contain confidential information. Sometimes, confidential data values are intrinsic to their problem, but often the problem could still be identified or resolved with innocuous data values that preserve some of the structure of the confidential data. Or, the confidential values are in variables that are unrelated to the problem.

While techniques for masking or disguising data exist, they are often complex or proprietary. In this paper, I describe a very simple macro, REVALUE, that can change the values in a SAS data set. REVALUE preserves some of the structure of the original data by ensuring that for a given variable, observations with the same real value have the same replacement value, and if possible, observations with a different real value have a different replacement value. REVALUE allows the user to specify the variables to change and whether to order the replacement values for each variable by the sort order of the real values or by observation order.

In this paper, I will discuss the REVALUE macro in detail, and provide a copy of the macro.

REVALUE MACRO: OBJECTIVES

1. To the extent possible, preserve the distinctness of data values within a variable.

Within a variable, the following is true.

- All observations with the same real value will have the same replacement value.
- To the extent possible, all observations with non-matching values will have non-matching replacement values. This is always true for numeric variables, and is true for character variables unless there are a very large number of distinct values.

2. Do not repeat replacement values across numeric variables.

If the value of the BASE= parameter is sufficiently large, numeric replacement variables are not repeated, as described in the section “Uniqueness of numeric replacement values” below.

3. Allow the user to choose one of two algorithms to mask values.

The user can select one of two algorithms for the variables to be masked.

- Order by value.
    - All observations with the lowest value get the lowest replacement value.
    - All observations with the second lowest value get the second lowest replacement value.
    - All observations with the third lowest value get the third lowest replacement value.
    - etc.
- Order by observation.
    - The first observation and all other observations with that value get the lowest replacement value.

- All observations with the second distinct value (the value in the second observation if it differs from first observation, else the value in the third observation if it differs from the first observation, etc.) get the second lowest replacement value.
- All observations with the third distinct value get the third lowest replacement value.
- etc.

4. Allow the user to specify the variables to be masked.

All variables in a data set can be masked, or the variables to mask can be limited by type (numeric or character) or with a list of variables to mask or omit. Variables that are not masked are written to the output data set unchanged.

THE GORY DETAILS: HOW VARIABLES ARE MASKED

1. How numeric variables are masked.

Macro parameter BASE, which defaults to 1,000,000, is a starting amount used to offset all numeric replacement values. Replacement values are calculated by (&BASE**n*) + *j* where

n is the nth numeric variable

j is the jth lowest value for Order by value or the jth distinct value for Order by observation.

The following table shows how the first few observations of the first few numeric variables are masked.

Numeric masking for Order by value

First variable to mask	General rule	Example for &BASE=1,0000,000
Every occurrence of the lowest value	&BASE+1	1,000,001
Every occurrence of the second lowest value	&BASE+2	1,000,002
Every occurrence of the third lowest value	&BASE+3	1,000,003

Second variable to mask	General rule	Example for &BASE=1,0000,000
Every occurrence of the lowest value	(&BASE*2)+1	2,000,001
Every occurrence of the second lowest value	(&BASE*2)+2	2,000,002
Every occurrence of the third lowest value	(&BASE*2)+3	2,000,003

Numeric masking for Order by observation

First variable to mask	General rule	Example for &BASE=1,0000,000
Every occurrence of the value in the first observation	&BASE+1	1,000,001
Every occurrence of the second distinct value	&BASE+2	1,000,002
Every occurrence of the third distinct value	&BASE+3	1,000,003

Second variable to mask	General rule	Example for &BASE=1,0000,000
Every occurrence of the value in the first observation	(&BASE*2)+1	2,000,001
Every occurrence of the second distinct value	(&BASE*2)+2	2,000,002
Every occurrence of the third distinct value	(&BASE*2)+3	2,000,003

Second distinct value means the value in the second observation if it differs from first observation, else the value in the third observation if it differs from the first observation, etc.

Uniqueness of numeric replacement values

Replacement values are unique within a variable.

If the value of BASE is sufficiently large, replacement values are unique across the entire data set. For example, if BASE=1000, then

- For the first numeric variable, the first replacement value is 1001, the second replacement value is 1002, etc.
- For the second numeric variable, the first replacement value is 2001, the second replacement value is 2002, etc.
- Numeric replacement values will be unique across the entire data set if no numeric variable has more than 1000 distinct values.

2. How character variables are masked.

1. The length of all replacement values for a variable is the longest length of the original values of that variable.

2. Replacement values range from

- a-z in the first character
- a-z 0-9 in all other characters

For example, if a variable's maximum length is 3, replacement values range from aaa to z99 (aaa, aab, aac, …) and the number of distinct replacement values is

26 * 36 * 36 = 33696

More generally, the number of distinct replacement values is

26 * (36** (variable_length-1))

If a character variable has more distinct values than the number of replacement values, replacement values are reused starting from the beginning value (all a's) and a note is printed to the SAS log. For example, if the maximum length is 3, there are 33696 distinct replacement values, and both the first and 33697th distinct values are assigned aaa.

3. All character variables with the same maximum length use the same replacement values, unlike numeric variables where the BASE parameter causes replacement values to differ for each variable,

4. Character variables are masked as follows, using variables with maximum length 3 to illustrate.

Order by value

Any variable whose maximum length = 3 gets these values:

- aaa for every occurrence of lowest value
- aab for every occurrence of second lowest value
- aac for every occurrence of third lowest value

- etc.

Order by observation

Any variable whose maximum length = 3 gets these values:

- aaa for every occurrence of the value in the first observation
- aab for every occurrence of the second distinct value (the value in the second observation if it differs from first observation, else the value in the third observation if it differs from the first observation, etc.)
- aac for every occurrence of the third distinct value
- etc.

Alternate approaches

Two alternate approaches to character variable masking were considered.

- Make the replacement string length equal to the variable's length. For example, if a character variable's length was 50 but the longest value was 10 characters, replacement values would be 50 characters instead of 10 characters.

  I decided against this approach because it would make the resulting data set harder to read and increase processing time, for little or no obvious benefit.

- Make the replacement string length equal to the length of the variable in each observation. Separate ascending ordering would take place for values of differing lengths. That is, for a given variable, the 4-byte character values (as ordered by either value or observation) would be replaced with aaaa, aaab, aaac, etc., and 5-byte character values would be replaced by aaaaa, aaaab, aaaac, etc.

  I decided against this approach because it would increase the complexity of the code and increase processing time.

EXAMPLE FOR A SMALL DATA SET

Here are the original and replacement values for a data set containing two numeric variables, NUM1 and NUM2, and one character variable, CHAR1. BASE=1000.

Original values			Order by value			Order by observation		
NUM1	NUM2	CHAR1	NUM1	NUM2	CHAR1	NUM1	NUM2	CHAR1
6	20	New York	1002	2002	aaaaaaac	1001	2001	aaaaaaaa
6	50	Iowa	1002	2004	aaaaaaaa	1001	2002	aaaaaaab
33	50	Iowa	1003	2004	aaaaaaaa	1002	2002	aaaaaaab
44	50	Ohio	1004	2004	aaaaaaad	1003	2002	aaaaaaac
33	20	Iowa	1003	2002	aaaaaaaa	1002	2001	aaaaaaab
5	40	Maine	1001	2003	aaaaaaab	1004	2003	aaaaaaad
6	10	Iowa	1002	2001	aaaaaaaa	1001	2004	aaaaaaab
5	10	New York	1001	2001	aaaaaaac	1004	2004	aaaaaaaa
5	10	Maine	1001	2001	aaaaaaab	1004	2004	aaaaaaad
6	20	Iowa	1002	2002	aaaaaaaa	1001	2001	aaaaaaab

MACRO REVALUE

Macro REVALUE is called as follows.

%REVALUE(data=*input-dataset*, out=*output-dataset*<,options>);

Required arguments

data=*input-dataset*
The input data set.

out=*output-dataset*
The output data set.

Options

type=num | numeric | char | character | all
The type of variables to be masked, one of the following
num or numeric for numeric variables only
char or character for character variables only
all for both numeric and character variables
Default is all.
TYPE= supersedes other selection criteria (VAR= and OMIT=).

order=value | observation
The ordering method for the replacement values.
Default is value.

base=*numeric-value*
The starting amount used as an offset to all numeric replacement values.
Default is 1,000,000.

var=*variable-list*
A space separated list of variables to mask.
You cannot specify both VAR= and OMIT=.
If not specified, mask all variables except as limited by OMIT= or TYPE=.
Variables in the list that conflict with TYPE= will not be masked.

omit=*variable-list*
A space separated list of variables to not mask.
You cannot specify both VAR= and OMIT=.
If not specified, mask all variables except as limited by VAR= or TYPE=.

Examples

1. Mask all variables in data set ONE and store the result in data set TWO.

```
%revalue(data=one, out=two);
```

2. Mask all numeric variables in data set ONE and store the result in data set TWO.

```
%revalue(data=one, out=two, type=numeric);
```

3. Mask all numeric variables in data set ONE except DATE1 and DATE2 and store the result in data set TWO.

```
%revalue(data=one, out=two, type=numeric, omit=date1 date2);
```

4. Mask all numeric variables in data set ONE except DATE1 and DATE2 and store the result in data set TWO. Use a BASE of 100 instead of 1,000,000.

```
%revalue(data=one, out=two, type=numeric, omit=date1 date2, base=100);
```

5. Mask the numeric variable BANKID in data set ONE and store the result in data set TWO. Use a BASE of 100 instead of 1,000,000. The character variable BANKNAME is ignored because TYPE=NUMERIC.

```
%revalue(data=one, out=two, type=numeric, var=bankid bankname, base=100);
```

Notes

1. Notes about variable selection.

- The TYPE parameter supersedes the VAR and OMIT parameters. For example, if VAR= includes character variables but TYPE=num, the character variables are not processed.

- A SAS date value is a good candidate to omit with OMIT= or VAR=.

2. The SEPSTR macro, in the Board's Linux SAS macro library, converts a character string of words into a character string where each word is surrounded by specified characters. It can be called in a DATA step, macro, or open code. It was written by Heidi Markovitz. This macro is included in Appendix 2.

MACRO REVALUE: SELECTED CODING NOTES

Macro REVALUE is included in Appendix 1. It is largely self-explanatory, but in addition to comments within the code, this section contains more detailed notes about a few code segments. The code is logically grouped into six sections. The notes in this section identify which section of the code is being referenced, and where necessary, a comment was added to the code for easy lookup.

1. Adding an extra variable for order by observation, in Section three.

For order by observation, I must sort the values of the current variable but then restore the original order. This requires an extra DATA step at the beginning that adds variable _N to the input data set, equal to the value of _N_.

2. Numeric replacement values, in Section four.

For each numeric variable, the following is done.

- Create a "cntlin data set" that maps the distinct values to an ascending integer number where the first value = 1, the second value = 2, etc. The ordering of the distinct values is by value or observation, based on the value of the ORDER= parameter.

- Supply the cntlin data sets to PROC FORMAT to create a numeric format for each numeric variable.

- Order by value and order by observation are coded separately for numeric variables.

3. Character replacement values, in Section five (right after the comment that says “See note 3 in the paper”).

For each character variable, the following is done.

- Create a "cntlin data set" that maps the distinct values to an ascending character string. Assuming the maximum length of that variable is 3, the first value = "aaa", the second value = "aab", etc. The ordering of the distinct values is by value or observation, based on the value of the ORDER= parameter.

- Supply the cntlin data sets to PROC FORMAT to create a character format for each character variable.

4. Generating the replacement character variables, in Section five (right after the comment that says “See note 4 in the paper”).

For character variables, the values for the first character, a-z, are in temporary character array CHAR_1, and the values for all other characters, a-z 0-9, are in temporary character array CHAR_ALLBUT1.

```
array char_1 (26) $1 _temporary_
    ("a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m"
     "n" "o" "p" "q" "r" "s" "t" "u" "v" "w" "x" "y" "z");
array char_allbut1 (36) $1 _temporary_
    ("a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m"
     "n" "o" "p" "q" "r" "s" "t" "u" "v" "w" "x" "y" "z"
     "0" "1" "2" "3" "4" "5" "6" "7" "8" "9");
```

Temporary array CHAR_COUNTER contains the indexes for the replacement values. It is initialized to all 1's, so that the first replacement value is all a's.

- CHAR_COUNTER(1) can range from 1 to 26 and is an index for the first replacement character, in CHAR_1.

- CHAR_COUNTER(2), CHAR_COUNTER(3), etc. can range from 1 to 36 and are indexes for all other replacement characters, in CHAR_ALLBUT1.

- CHAR_COUNTER's size is the maximum length for the current variable, which is the size of the replacement string.

```
array char_counter (&current_max_length) _temporary_ (&current_max_length*1);
```

The current replacement value is generated with the CATS function. For example, if the replacement variable length (&CURRENT_MAX_LENGTH) is 3 and we are processing the third distinct value, then CHAR_COUNTER(1)=1, CHAR_COUNTER(2)=1, and CHAR_COUNTER(3)=3 and LABEL will be "aac".

```
label = cats(char_1(char_counter(1))
```

```
    %do i=2 %to &current_max_length;
      , char_allbut1(char_counter(&i))
    %end;
  );
```

I considered an alternate approach to the CATS function: storing the replacement value in a character variable and incrementing the replacement value with the SUBSTR function. For example, if the replacement value had length 3 and we are processing the third distinct value, the character variable's value would be "aac".

To benchmark, I tested a variable with 999,999 distinct values and a length of 6, and there was no meaningful time difference.

5. Replacing the values, in Section 6.

In the final step, the formats are used to change the values of the relevant variables.

CONCLUSION

This paper reviewed a very simple macro that can mask confidential values in a SAS data set. This topic is an area of ongoing interest to me, and I welcome suggestions for improving or extending the macro.

REFERENCES

SAS Institute Inc. (2012), "*Base SAS 9.3 Procedures Guide, Second Edition*," Cary, NC: SAS Institute Inc.

SAS Institute Inc. (2011), "SAS 9.3 Language Reference by Name, Product, and Category," Cary, NC: SAS Institute Inc.

SAS Institute Inc. (2011), "SAS 9.3 Macro Language: Reference," Cary, NC: SAS Institute Inc.

ACKNOWLEDGMENTS

The following people contributed extensively to the development of this paper: Heidi Markovitz and Donna Hill at the Federal Reserve Board. Their support is greatly appreciated.

CONTACT INFORMATION

For more information, contact the author, Bruce Gilsen, by mail at Federal Reserve Board, Mail Stop N-122, Washington, DC 20551; by e-mail at bruce.gilsen@frb.gov; or by phone at 202-452-2494.

TRADEMARK INFORMATION

APPENDIX 1

This appendix contains the REVALUE macro.

```
%macro revalue (data=, out=, type=all, order=value, base=1000000, var=, omit=);
  %local omit_or_var allname currentname j dotlocation libnam memnam numeric_vars
         num_numeric_vars all_numeric_formats max_char_lengths longest_char_length
         current_max_length character_vars num_character_vars all_character_formats;

  /*****************************************************************************
   Section One.  Error checking.
  *****************************************************************************/

    /* Do some error checking first */
  %if &data= %then %do;
    %put Macro revalue aborts: no input data set provided;
    %return;
  %end;
  %else %if &out= %then %do;
    %put Macro revalue aborts: no output data set provided;
    %return;
  %end;
  %if &var ne and &omit ne %then %do;
    %put Macro revalue aborts: cannot specify both VAR= and OMIT=;
    %return;
  %end;
  %if &order ne value and &order ne observation %then %do;
    %put Macro revalue aborts: invalid value for ORDER=;
    %return;
  %end;
  %if &type = numeric %then %do;
     %let type=num;
  %end;
  %else %if &type = character %then %do;
     %let type=char;
  %end;
  %else %if &type ne num and &type ne char and &type ne all %then %do;
    %put Macro revalue aborts: invalid value for TYPE=;
    %return;
  %end;

  /*****************************************************************************
   Section Two.  Initialize variable list (if any) and data set and library name.
  *****************************************************************************/

  %if &omit ne or &var ne %then %do;
    /* If names to omit or select specified, build space
       separated upper case quoted list of names */
```

```
 %let allname = %sepstr(%upcase(&var &omit),separator=%str());
%end; /* of %if omit.... */

  /* If 2 Section data set name, split it at the ".", otherwise libname is WORK */
%let dotlocation=%index(&data,%str(.));
%if &dotlocation=0 %then %do;
   %let libnam=WORK;
   %let memnam=%upcase(&data);
%end;
%else %do;
   %let libnam=%upcase(%substr(&data,1,%eval(&dotlocation-1)));
   %let memnam=%upcase(%substr(&data,%eval(&dotlocation+1)));
%end;

/*****************************************************************************
 Section Three.  Add extra variable _N to input data set with observation number
 if user requested Order by Observation.
*****************************************************************************/

  /* If order is by observation, create copy of input data set and add variable
     _N containing observation number.
     Both numeric and character variables use this data set, so do this only once
     here, outside of the loops for character and numeric variables. */
%if &order eq observation %then %do;
  data _cntl;
    set &data;
    _n=_n_;
  run;
%end;

/*****************************************************************************
 Section Four.  Process numeric variables.
*****************************************************************************/

%if &type ne char %then %do; /* process numeric variables */

    /* Create space separated list of numeric variables to modify */
  proc sql noprint;
    select name
    into :numeric_vars separated by ' '
    from dictionary.columns
    where libname="&libnam"
    and memname="&memnam"
    and type = "num"
  %if &omit ne %then %do; /* omit variables if requested */
    and upcase(name) not in (%unquote(&allname))
  %end;
  %else %if &var ne %then %do; /* select variables if requested */
```

```
  and upcase(name) in (%unquote(&allname))
%end;
  ;
%let num_numeric_vars=&sqlobs; /* how many selected */
quit;

  /* If there are any numeric variables to process, do it */
%if &num_numeric_vars ne 0 %then %do;

  %let all_numeric_formats=; /* format for each variable to convert */
  %do j = 1 %to &num_numeric_vars;
    %let currentname = %scan(&numeric_vars, &j, %str( ));
    %let all_numeric_formats= &all_numeric_formats "f&currentname.f";

    %if &order eq value %then %do;      /* order by value */

        /* For jth variable, create sorted data set with distinct values.  Add
           variables needed for a "cntlin data set" that maps the values to an
           ascending integer number: 1st value in dataset=1, 2nd value = 2, etc.,
           and create a format.
           Append f to variable name to make format name to ensure last character
           of format name is a letter, as required.
        */
      proc sort data=&data(keep=&currentname) nodupkey out=cntl;
        by &currentname;
      run;
      data cntl(rename=(&currentname=start));
        retain fmtname "f&currentname.f"; /* name of format */
        set cntl end=endval;
        label = _N_;
        output; /* write out value */
        if endval then do;
          /* Add extra observation to cntlin data set that maps all
             values of &NUMERIC_VARS not in format table to missing. */
          hlo = 'O';
          label  = . ;
          output ;
        end;  /* of if endval then do */
      run;

        /* Create a format that maps jth value in macro variable NUMERIC_VARS
           to J */
      proc format cntlin = cntl ;
      run;

    %end; /* of %if &order eq value %then %do; */

    %else %do;                /* order by observation */
```

```
  /* For jth variable, create sorted data set with distinct values that is
     ordered as per the original data set.  Add variables needed for a
     "cntlin data set" that maps the values to an ascending integer number:
     1st value in dataset=1, 2nd value = 2, etc., and create a format.
     Append f to variable name to make format name to ensure last character
     of format name is a letter, as required.
     Here is an example for a typical variable.
     (1) Original data                  (2) First Sort result
       Variable values  _N              Variable values      _N
         9               1                   4               5
        13               2                   9               1
        13               3                   9               4
         9               4                  13               2
         4               5                  13               3

     (3) DATA step with FIRST. result    (4) Second sort result
       Variable values  _N               Variable values     _N
         4               5                   9               1
         9               1                  13               2
        13               2                   4               5

     (5) Then store _NLABEL, ascending number starting at 1, as the label,
     so that values are mapped as follows:
      START            LABEL
        9                1
       13                2
        4                3
    */
proc sort data=_cntl out=cntl;
   by &currentname _n;
run;
data cntl;
  set cntl;
  by &currentname _n;
  if first.&currentname;
run;
proc sort data=cntl out=cntl;
   by _n;
run;

data cntl(rename=&currentname=start);
  retain _nlabel 1;
  retain fmtname "f&currentname.f"; /* name of format */
  set cntl end=endval;
  drop _nlabel;
  label = _nlabel;
  output; /* write out value */
```

```
          _nlabel+1;
          if endval then do;
            /* Add extra observation to cntlin data set that maps all
               values of &NUMERIC_VARS not in format table to missing. */
             hlo = 'O';
             label  = . ;
             output ;
          end;  /* of if endval then do */
        run;

          /* Create a format that maps jth value in macro variable
             NUMERIC_VARS to J */
        proc format cntlin = cntl ;
        run;

      %end; /* of %else %do to order by observation */
    %end; /* of %do j = 1 %to &num_numeric_vars; */
  %end; /* of %if &num_numeric_vars ne 0 %then %do; */
%end; /* of %if &type ne char %then %do; */
%else %do;
  %let num_numeric_vars = 0; /* no numeric variables to convert */
%end;

/*******************************************************************************
 Section Five.  Process character variables.
*******************************************************************************/

%if &type ne num %then %do; /* process character variables */

    /* Create space separated list of character variables to modify */
  proc sql noprint;
    select name
    into :character_vars separated by ' '
    from dictionary.columns
    where libname="&libnam"
    and memname="&memnam"
    and type = "char"
  %if &omit ne %then %do; /* omit variables if requested */
    and upcase(name) not in (%unquote(&allname))
  %end;
  %else %if &var ne %then %do; /* select variables if requested */
    and upcase(name) in (%unquote(&allname))
  %end;
    ;
  %let num_character_vars=&sqlobs; /* how many selected */
  quit;

    /* If there are any character variables to process, do it */
```

```
%if &num_character_vars ne 0 %then %do;

    /* MAX_CHAR_LENGTHS is space separated macro variable with maximum length of
       each character variable, used as initial values in a temporary array.
       Macro variable LONGEST_CHAR_LENGTH has maximum length of any variable. */

  data _null_;
    set one (keep= &character_vars) end=last;
    drop i;
    array all_chars (*) &character_vars;
    array all_chars_max (&num_character_vars) _temporary_ (&num_character_vars*1);
    do i = 1 to dim(all_chars);
      if length(all_chars(i)) gt all_chars_max(i)
         then all_chars_max(i) = length(all_chars(i));
    end;

    if last then do;
      /* Maximum length of &MAX_CHAR_LENGTH is as follows:
         Assume maximum variable length = 32k = 5 digits.
         If 1000 variables, max size of macro variable is
         (5 * 1000) + 999 spaces between each value = 5999. */
      call symput('max_char_lengths',catx(" ",of all_chars_max(*)));
      /* &LONGEST_CHAR_LENGTH is longest single character variable length,
         used to determine size of array with indices for the replacement values.
         MAX function requires at least 2 arguments and ALL_CHARS_MAX array could
         have size 1, so include . in MAX function to avoid error.  . sorts lowest
         so it does not affect the answer. */
      call symput('longest_char_length',
           compress(put(max(of all_chars_max(*),.),5.)));
    end;
  run;

  %let all_character_formats=; /* format for each variable to convert */
  %do j = 1 %to &num_character_vars;
    %let currentname = %scan(&character_vars, &j, %str( ));
      /* longest length for current variable */
    %let current_max_length = %scan(&max_char_lengths, &j, %str( ));
    %let all_character_formats= &all_character_formats "$&currentname.f";

                                                  /**** See note 3 in the paper ****/
       /* For jth variable, create sorted data set with distinct values.  Add
          variables needed for a "cntlin data set" that maps the values to an
          ascending character string, e.g., aaa aab aac ... if longest string
          length for the variable is 3.
          Create a format from it, append f to name of variable to make format
          name to ensure last character of format name is a letter, as required.
          Order the original values in either ascending order or observation
          order based on value of &ORDER.
```

```
    */

%if &order eq value %then %do;      /* order by value */
  proc sort data=&data(keep=&currentname) nodupkey out=cntl;
    by &currentname;
  run;
%end;   /* of if &order eq value %then %do; */

%else %do;                        /* order by observation */

     /* Example of ordering data prior to creating the
        format for a typical variable when ORDER=OBSERVATION.

        (1)Original data                    (2) First sort result
          Variable values  _N             Variable values     _N
          b                 1                a                 5
          cc                2                b                 1
          cc                3                b                 4
          b                 4                cc                2
          a                 5                cc                3

        (3) DATA step with FIRST. result    (4) Second sort result
          Variable values  _N                Variable values  _N
          a                 5                b                 1
          b                 1                cc                2
          cc                2                a                 5

        (5) Then store the label as ascending strings, aa ab ac, so that
        values are mapped as follows:
         START           LABEL
           b               aa
          cc               ab
           a               ac
      */
  proc sort data=_cntl out=cntl;
     by &currentname _n;
  run;
  data cntl;
    set cntl;
    by &currentname _n;
    if first.&currentname;
  run;
  proc sort data=cntl out=cntl;
     by _n;
  run;
%end;   /* of %else %do; for order by observation */

  /* Now create the format for both cases: order by value
```

```
    and order by observation */

                                                /**** See note 4 in the paper ****/
data cntl(rename=(&currentname=start));
  retain fmtname "$&currentname.f"; /* name of format */
    /* replacement string length */
  retain current_max_length &current_max_length;
  set cntl end=endval;

    /* 1st character is a-z, others are a-z 0-9 */
  array char_1 (26) $1 _temporary_
      ("a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m"
       "n" "o" "p" "q" "r" "s" "t" "u" "v" "w" "x" "y" "z")
       ;
  array char_allbut1 (36) $1 _temporary_
      ("a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m"
       "n" "o" "p" "q" "r" "s" "t" "u" "v" "w" "x" "y" "z"
       "0" "1" "2" "3" "4" "5" "6" "7" "8" "9")
       ;
    /* Indexes for replacement value, initially all 1,
       i.e., 1st value of temporary arrays is all "a"s */
  array char_counter (&current_max_length) _temporary_
      (&current_max_length*1);
    /* Need LENGTH statement even though length is same in
       all observations because CATS defaults to length 200 */
  length label $&current_max_length;

  label = cats(char_1(char_counter(1))
      %do i=2 %to &current_max_length;
        , char_allbut1(char_counter(&i))
      %end;
    );
  output; /* write out value */

    /* Increment replacement character index variable(s).  Add 1 to lowest
       order index value but if that exceeds the number of characters, also
       update earlier values as needed.  Since 1st index value can only go to
       26, that serves to stop the DO WHILE loop.
       Examples, assuming CURRENT_MAX_LENGTH is 3:
       char_counter (before)    char_counter (after)
       10 10 10                 10 10 11
       10 30 36                 10 31 1
       10 36 36                 11 1 1
       26 36 36                 1  1 1 (and print note) */
  char_counter(current_max_length) + 1;
  char_index=current_max_length;
  do while (char_counter(char_index) = 37);
    char_counter(char_index) = 1;
```

```
          char_index=char_index-1;
          char_counter(char_index) + 1;
        end;
        if char_counter(1) = 27 then do;
          /* no more unique values to revalue to */
          char_counter(1) = 1;
          put "NOTE: No more unique replacement values for &currentname,
               reusing previous values";
        end;

        if endval then do;
          /* Add extra observation to cntlin data set that maps all
             values of &CHARACTER_VARS not in format table to missing. */
           hlo = 'O';
           label  = "" ;
           output ;
        end;  /* of if endval then do */
      run;

        /* Create a format that maps jth value in macro variable
           CHARACTER_VARS to J */
      proc format cntlin = cntl ;
      run;

    %end; /* of %do j = 1 %to &num_character_vars; */
  %end; /* of  %if &num_character_vars ne 0 %then %do; */
%end; /* of %if &type ne num %then %do; */
%else %do;
  %let num_character_vars = 0; /* no character variables to convert */
%end;

/*******************************************************************************
 Section Six.  Read the input data set and mask the values.
*******************************************************************************/

data &out;
  set &data;
  drop i;

    /* If there are numeric variables to convert:
       array NUM_TO_CONVERT has numeric variables to convert.
       array ALLNUMERICFORMATS has formats for the corresponding variables in
       NUM_TO_CONVERT.
    */
  %if &num_numeric_vars ne 0 %then %do;
    array num_to_convert (*) &numeric_vars;
    array allnumericformats (&num_numeric_vars) $32 _temporary_
          (&all_numeric_formats);
```

```
      do i = 1 to dim(num_to_convert);
        num_to_convert(i) =
           (&base*i)+input(putn(num_to_convert(i),allnumericformats(i)),best.);
      end;
    %end;

      /* If there are character variables to convert:
         array CHAR_TO_CONVERT has character variables to convert.
         array ALLCHARACTERFORMATS has formats for the corresponding variables in
         CHAR_TO_CONVERT.
      */
    %if &num_character_vars ne 0 %then %do;
      array char_to_convert (*) &character_vars;
      array allcharacterformats (&num_character_vars) $32 _temporary_
            (&all_character_formats);
      do i = 1 to dim(char_to_convert);
        char_to_convert(i) =
           putc(char_to_convert(i),allcharacterformats(i));
      end;
    %end;
run;

%mend revalue;
```

APPENDIX 2

This appendix lists and documents the SEPSTR macro, which is called by the REVALUE macro.

%SEPSTR(input-string<,options>);

Required arguments

input-string

The character string to convert.

Options

prefix=*prefixarg*

The character(s) to place before each word in the output character string. See suffixarg for details.

Default: double quote.

suffix=*suffixarg*

The character(s) to place after each word in the output character string. Since *prefixarg* and *suffixarg* both default to double quotes, if either is set to another value, you must also set the other. If only one of the arguments is needed, set the other to a blank, %STR().

Default: double quote.

separator=*separatorarg*

One or more characters to place between each word in the output character string. The value followed by a single space will follow every word except the last word.

Default: comma.

```
%MACRO sepstr(input_string, prefix=%STR(%"),suffix=%STR(%"),separator=%STR(,));
   %LET x=1;
   %LET out_string=;
   %DO %WHILE (%SCAN(%UNQUOTE(&input_string), &x) NE %STR());
      %LET thisword = %SCAN(%QUOTE(&input_string), &x);
      %IF &x > 1
          %THEN %LET out_string = %QUOTE(&out_string&separator);
      %LET out_string = %UNQUOTE(&out_string) &prefix&thisword&suffix;
      %LET x = %EVAL(&x + 1);
   %END;
   &out_string
%MEND sepstr;
```