List Processing Macro Call-Macro

Ronald J. Fehd, Stakana Analytics, Atlanta, GA, USA

Abstract

Description : This paper provides an explanation of the general-purpose list processing routine Call-Macro. The parameters of this macro are a data set name and a macro name. It reads each row of the date set and generates a call to the macro named in its parameter.

Purpose : The purpose of this paper is to show how to use the macro function `sysfunc` with the SCL data set functions that `open` a data set, read and process each variable in an observation, and then `close` the data set.

Audience : intermediate macro users, applications programmers

Programs : in this paper are available in Fehd [3] `http://www.sascommunity.org/wiki/Macro_CallMacr`

Keywords : module, routine, subroutine, macro function `sysfunc`,
data set functions: `attrn, exist, open, close, fetchobs, getvarc, getvarn, varname, vartype`

Contents

Introduction

Writing a list processing application in SAS® software consists of three tasks. The first task is to prepare a subroutine which does the process or produces the output, the product. Next is to prepare a list, a control data set, in which each row is a set of values of the parameters of the subroutine. Finally, read the control data set and generate calls to the macro containing the output subroutine.

Macro `callmacr.sas` is a list-processing routine which reads a control data set, converts the variables and values in each observation to named macro parameters and then calls the macro subroutine.

This is the list of parameters.

- `data`, one- or two-level data set name
- `macro_name`, name of macro to call
- `macro_parms`, additional parameters for the called macro
- `hex16`, convert numerics to hexadecimal;
  used to pass real numbers accurately across step boundaries

This routine solves several problems of list processing.

- is a general solution which eliminates the need to write customized code for each list processing application
- hides the complexity of processing any list
- converts all numbers to hexadecimal to preserve accuracy when passing numbers across step boundaries
- eliminates using a data step to read the list
- contains testing-ware statements to aid in unit and integration tests

This introduction has the following topics.

- list definition
- program definitions
- developing subroutines
- making lists
- processing a list

What is a List?

A list is defined in three differents set of vocabulary.

natural : In natural language a list contains items. Examples are a shopping list, where items are names of objects to purchase. A cell-phone has a contact list which has entries; the items may be character or telephone numbers; some items may be blank.

theory : The theoretical definition of a list is recursive. A list contains items, which may be referred to as elements or atoms. An item may be an atom, or a list. The list may or may not be ordered. Access is sequential.

SAS : In this article a data set is used as a list. Some of the elements of a data set are its attributes, such as the label, number of observations (rows) and number of variables (columns). Each row of the two-dimensional matrix is a set of parameters for some other process.

control data set : The format procedure statement has an option named `cntlin` the value of which is `input-control-data-set`. A data set with a specific data structure can be used as input to the format procedure to create a format, instead of a `format` statement.

A list is a control data set; the variable names in the list match the names of the parameters of the macro processing subroutine.

Continued on next page.

Program Definitions

Programs can be organized and grouped into several categories, both theoretical and practical.

acronym : All programs have the following list of items or actions.

- **H**ierarchical
- **I**nput
- **P**rocess
- **O**utput

This phrase has been reduced to the mnemonic HIPO.

theory : Within the hierarchy of program organization a program may have several levels of calls to other programs. The number of levels can be theoretically reduced to exactly three: top, middle, bottom.

- module calls modules, routines and subroutines
- routine calls routines and subroutines
- subroutine does not call other programs

practice : These are the types of SAS programs.

- %include
- %include with parameters
- macro produces statements and steps
- macro function produces tokens within a statement, i.e. it does not produce a semicolon

---

Developing Subroutines

Processing subroutines are written as independent programs so that they can be unit tested. Their purpose is to process one item of a list. The first consideration in developing a subroutine is to consider and eliminate the possibility that by-group processing which processes all items of a list as an entity is the solution. A list processing subroutine processes exactly one item of the list, whether that item is a text file, a data set, a variable, or a date interval.

This is a short check list for subroutine development.

- parameters must agree with the variables in the list, the control data set
- style guide with naming conventions
- write testing-aware code

Continued on next page.

Making Lists

Any data set produced by a procedure is a candidate for reuse as a list, a control data set.

- `contents`
- summarization procedures `freq, summary`
- `sql dictionary.tables, dictionary.columns`
- data steps to read list of files or folders

The examples shown in the testing section on page 8, use this standardized `proc freq` output data set.

call-macro-test-class.sas

```
4  PROC freq data   = sashelp.class;
5            tables   sex
6                   / list missing
7               out = freq_class_sex
8           (rename = (sex = value));
9            title3   'frequency output';
10 run;*necessary;
11 proc print data = &syslast  noobs;
12           title3  'list :: control data set';
```

Task: Processing Lists

The task is to encapsulate and hide the complexity of reading a list, a control data set, and processing each observation to produce calls to a macro.

This is an example of standardized output from the frequency procedure; the code is shown above.

```
1  list :: control data set
2  value  COUNT  PERCENT
3     F       9   47.3684
4     M      10   52.6316
```

The task of this article is to show how to convert the above list to these macro calls.

```
1  %*process;
2  %callmacr(data       = freq_class_sex
3           ,macro_name = proc_this     )
4  %*tokens produced;
5  %proc_this(value=F,count= 9,percent=47.3684)
6  %proc_this(value=M,count=10,percent=52.6316)
```

These are the goals for the routine.

- process any data set, without regard to number or type of variables
- eliminate using data step to read list
- preserve accuracy of numeric values passed across step boundaries

---

Algorithm

Overview

This section provides an overview of the algorithm of the macro.

1. functions
2. initializations
3. assertions
4. loops
5. assemble string: varname=value
6. call macro
7. print time used note

Functions

This is the list of macro and SCL functions used in the macro.

macro :
- `%eval`, evaluate an expression and return integer result
- `%nrstr`, no-rescan-string: do not parse and expand macro variable special characters ampersand (&)and percent (%)
- `%unquote`, parse string and expand macro references
- `%sysfunc` is used with the SCL functions

SCL : SAS Component Language (SCL)
- `exist`, `open`, `close` of data set
- `attrn` to fetch nobs and nvars of data set
- `fetchobs` to get values of all variables in an observation
- `varname`, `varnum`, and `vartype` of each variable
- `putn` used to display numeric values
- `getvarc`, `getvarn` are used in assignment statements for each variable type

Continued on next page.

Initializations Two macro variables are reassigned values.

semicolon : During testing the tokens returned may be a statement which requires an ending semicolon. The macro variable `semicolon` is boolean; its default value is zero: no semicolon is produced. When the value of macro variable `macro_name` is the string `put note` then the value of `semicolon` is set to one.

```
%let semicolon = %eval(&semicolon
or %lowcase(%scan(&macro_name ,1,%str( ))) eq put);
```

testing : To avoid multiple tests of user-supplied true values such as `y, Y, yes, Yes` the logical expression `not(0 eq &testing)` recodes any value other than zero to one.

The default values of options `mprint` and `source2` are `nomprint` and `nosource2`. During testing the macro variable `testing` can be set to true (1) by turning on those options with the statement `options mprint source2`. Note the `getoption` function returns upper case.

```
%let testing=%eval(   not(0 eq &testing)
or(    %sysfunc(getoption(mprint )) eq MPRINT
   and %sysfunc(getoption(source2)) eq SOURCE2 ));
```

Assertions The macro expects two assertions in order to complete its process.

existence : Does the data set exist?

```
%if  not(%sysfunc(exist(&data))) %then %do;
     %put note: 0.1 &sysmacroname exit
                    not exist(&data);
     %return;
     %end;
```

not empty : Are there both observations and columns

```
%let dsid   = %sysfunc(open (&data      ));
%let n_obs  = %sysfunc(attrn(&dsid,nobs ));
%let n_vars = %sysfunc(attrn(&dsid,nvars));
%if  not &n_obs or not &n_vars %then %do;
     %put note: 0.2 &sysmacroname &data exit
                    obs=&n_obs vars=&n_vars;
     %goto close_exit;
     %end;
%*...;
%close_exit: %let rc = %sysfunc(close(&dsid));
```

Note: the logical expression is an example of De Morgan's *nand* which can also be written with parenthesis.
`not (&n_obs and &n_vars)`

Continued on next page.

Loops

Two loops read the two-dimensional matrix of the data set. For each variable the name is needed for use in the macro variable `list_parameters`. In order to get the variable type in (character, number) the variable number is required.

```
%do     row    = 1 %to &n_obs;
    %do column = 1 %to &n_vars;
        %let name = %sysfunc(varname(&dsid,&column));
        %let num  = %sysfunc(varnum (&dsid,&name  ));
        %let type = %sysfunc(vartype(&dsid,&num   ));
        %*...;
        %end;
    %*...;
    %end;
```

Assemble String

In each observation the variable names and values are concatenated in the macro variable `list_parameters`.

```
%*** intialize;
%let list_parms = ;
%*** in loop;
%*** append string: name=;
%let list_parameters = &list_parameters&name=;
%*** append string: value;
%let list_parameters = &list_parameters%left
        (%sysfunc(getvar&type(&dsid,&num)));
%**  append comma;
%if &column lt &n_vars %then
    %let list_parameters = &list_parms,;
```

At the end of the observation the string looks like this.

```
var_a=valu-a,var_b=valu-b,...,var_z=valu-z
```

Continued on next page.

Call Macro

A macro call consists of these items.

1. percent sign
2. macro name
3. open parenthesis
4. list of named parameters and values separated by commas
5. `close` parenthesis

```
%&macro_name(&list_parameters)
```

Note this is a set of tokens; it is not a statement and therefore does not require a semicolon after the closing parethesis.

Write Time Used

Two macro variables are used to capture time-start and time-end of the routine. The `putn` function is used to supply a run-time format. The difference between the start and end time is calculated and supplied as the argument to the `putn` function call.

```
%*initialization;
%let time_start = %sysfunc(datetime(),hex16);
%*...;
%let time_end   = %sysfunc(datetime(),hex16);
%put note: &sysmacroname used real time %sysfunc
           (putn(&time_end.x-&time_start.x,time12.3));
%mend callmacr;
```

The format `time12.3` displays the elapsed number of seconds with three decimal places: hh:mm:ss.sss.

Testing

Overview

This section presents three programs which are a minimum test of the macro.

- macro print subset
- testing program
- example module

Continued on next page.

Print Subset This is the prototype program from which the print-subset macro was developed.

```
proc print data  = sashelp.class
          (where = (sex = 'F'));
```

```
 /*    name: print_subset.sas
description: print subset of a data set
             list processing demonstration
purpose    : show how to
          1. determine variable type
          2. convert hex16 to decimal
          3. round off for display
note       : list is from contents and freq ***/
%macro print_subset
      (libname = sashelp /* contents */
      ,memname = class   /* contents */
      ,name    = sex     /* contents */
      ,value   = F       /* freq */
      ,count   = 1       /* freq */
      ,percent = 10      /* freq */
      ,testing = 0
      );
%let testing=%eval(   not(0 eq &testing)
     or(    %sysfunc(getoption(mprint))  eq MPRINT
        and %sysfunc(getoption(source2)) eq SOURCE2 ));
%if &testing %then %do;
    %put _local_;
    %end;
%let _dsid    = %sysfunc(open    (&libname..&memname));
%let _varnum  = %sysfunc(varnum  (&_dsid,&name     ));
%let _vartype = %sysfunc(vartype (&_dsid,&_varnum  ));
%let _label   = %sysfunc(varlabel(&_dsid,&_varnum  ));
%let _rc      = %sysfunc(close   (&_dsid           ));

PROC print data   = &libname..&memname
          (where  = (&name eq
          %if       &_vartype eq C %then "&value";
          %else %if &_vartype eq N %then  &value ;
     title2 "data    : &libname..&memname";
     title3 "subset  : &name eq &value";
     title4 "freq    : count: &count ";
     title5 "         percent: &percent";
     title6 count   real: %sysfunc(inputn(&count  ,hex16.));
     title7 percent real: %sysfunc(inputn(&percent,hex16.));
     title8 percent, 5.1: %sysfunc(round(%sysfunc
                                  (inputn(&percent,hex16.))
                                  ,.1));
     title9 "vartype : &name is &_vartype";
     title10 "varlabel: %left(&_label)";
run; title2;
%mend print_subset;
```

Continued on next page.

Testing Program This program is the unit test of the macro.

```
* name: call-macro-test-class.sas;
options mprint source2; * testing on;

PROC freq data   = sashelp.class;
          tables   sex
                 / list missing
             out = freq_class_sex
          (rename = (sex = value));
           title3   'frequency output';
run;*necessary;
proc print data = &syslast  noobs;
           title3  'list :: control data set';
%callmacr(data        = freq_class_sex
         ,macro_name  = put note:
         ,macro_parms = %nrstr(data=sashelp.class,name=sex)
         ,hex16       = 0
         )
%callmacr(data        = freq_class_sex
         ,macro_name  = print_subset
         ,macro_parms = %nrstr
         (libname=sashelp,memname=class,name=sex)
         )
```

list : The listing from `proc freq` shows the tables variable as `Sex`. This variable is renamed to `value` in the output data set

```
frequency output
                                Cumulative  Cumulative
Sex  Frequency  Percent   Frequency    Percent
-----------------------------------------------
F            9    47.37           9      47.37
M           10    52.63          19     100.00

list :: control data set
value  COUNT  PERCENT
   F       9   47.3684
   M      10   52.6316
```

log : Compare the printed values above with the notes written to the log.

```
11 %callmacr(data        = freq_class_sex
12          ,macro_name  = put note:
13          ,macro_parms =
            %nrstr(data=sashelp.class,name=sex)
14          ,hex16       = 0
15          )
note: 1 CALLMACR reading freq_class_sex obs=2 vars=3
...
note:(data=sashelp.class,name=sex,value=F
      ,count=9,percent=47.3684210526315)
```

Continued on next page.

macro log : This is the log from the call of macro `callmacr` which calls macro `print_subset`.

```
16 %callmacr(data        = freq_class_sex
17          ,macro_name  = print_subset
18          ,macro_parms = %nrstr
19          (libname=sashelp,memname=class,name=sex)
20          )
note: 1 CALLMACR reading freq_class_sex obs=2 vars=3
note: 2 testing: libname=sashelp,memname=class,name=sex
,value=F,count=4022000000000000,percent=4047AF286BCA1AE7
PRINT_SUBSET LIBNAME sashelp
PRINT_SUBSET MEMNAME class
PRINT_SUBSET NAME sex
PRINT_SUBSET VALUE F
PRINT_SUBSET COUNT   4022000000000000
PRINT_SUBSET PERCENT 4047AF286BCA1AE7
PRINT_SUBSET TESTING 1
```

macro output : This is the listing from the macro call which shows the hex16 vaues of variables `count` and `percent` and their decimal values.

Note `percent` has been rounded to format 5.1.

```
data    : sashelp.class
subset  : sex eq F
freq    : count: 4022000000000000
        percent: 4047AF286BCA1AE7
count   real: 9
percent real: 47.3684210526315
percent, 5.1: 47.4
vartype : sex is C
varlabel:

Obs Name    Sex  Age  Height  Weight

  2 Alice    F    13   56.5     84.0
  3 Barbara  F    13   65.3     98.0
...
 14 Mary     F    15   66.5    112.0
```

Continued on next page.

Example Module

This test program is an example of a module which can be easily modified for any other data set and variable.

```
 /*    name: call-macro-test-shoes.sas;
description: test program
    purpose: template for module
             for use of callmacr ******/
options mprint source2; * mautocomploc ;

%let in_lib   = sashelp;
%let in_data  = shoes;
%let in_var   = region;
%let out_list = freq_&in_data._&in_var;

PROC freq data   = &in_lib..&in_data;
          tables   &in_var
                 / list missing
             out = &out_list
         (rename = (&in_var = value));
run;*necessary;
%callmacr(data        = &out_list
         ,macro_name  = print_subset
         ,macro_parms = %nrstr
         (libname=&in_lib,memname=&in_data,name=&in_var)
         )
```

Program Listing callmacr.sas

```
 /*       name: <UNC>\SAS-site\macros\callmacr.sas
        author: Ronald J. Fehd  2013
                ------------------------------------------------------
Summary       : description  : call macro using
                               all values in data set row as parameters
                purpose      : provide generic method to call a macro
                               using list::control data set of parms
                ------------------------------------------------------
Contexts      : program group: list processing token generator
                program  type: routine
                SAS      type: macro function
                uses routines: macro named in the parameter macro_name
                ------------------------------------------------------
Specifications: input  : required: data, macro_name
                         optional: macro_parms, hex16
                process: assemble macro-call, call
                output : from macro_name
                ------------------------------------------------------
Parameters    : data        = one- or two-level data set name
               ,macro_name  = name of macro to call
               ,macro_name  = put :: default, for testing
               ,macro_parms = additional parameters for called macro
                              *-constraint-*: must be enclosed in nrstr:
               ,macro_parms = %nrstr(data=sashelp.class,var=Sex)
               ,hex16       = 1 :: default, convert numerics to hex16
                              used to pass real numbers accurately
                                  across step boundaries
               ,hex16       = 0 :: pass numerics as decimals
               ,semicolon   = 0 :: no semicolon after macro call
               ,semicolon   = 1 :: use when macro_name is a statement
                                   note: reset when macro_name=put
               ,testing     = 0 :: default, no extra messages in log
               ,testing     = 1 :: for testing, note: auto-reset when
                                   options mprint source2;
                ------------------------------------------------------
Bells,Whistles: writes real time used to log
                note: CALLMACR used real time  0:00:00.016
----------------------------------------------------------------------
```

```
Usage Example:
PROC freq data   = sashelp.class;
          tables   sex
                 / noprint
             out =      freq_class_sex;
run;*necessary;
%callmacr(data        = freq_class_sex
         ,macro_name  = put note:
         ,macro_parms = %nrstr(data=sashelp.class,var=sex)
         ,hex16       = 0)
log:
note:(data=sashelp.class,var=sex
     ,sex=F,count=9,percent=47.3684210526315)
-------------------------------------------------------------------------
NOTE CAREFULLY: character variables may contain special characters:
16  label = 'x&y';
17  output work.special_ampersand;
18  label = 'x,y';
19  output work.special_comma;
20  label = 'x%y';
21  output work.special_percent;
24  %callmacr(data = work.special_ampersand)
WARNING: Apparent symbolic reference Y not resolved.
(label=x&y)
28  %callmacr(data = work.special_comma)
ERROR: More positional parameters found than defined.
(label=)
29  %callmacr(data = work.special_percent)
WARNING: Apparent invocation of macro Y not resolved.
(label=x%y)
-------------------------------------------------------------------------
DateTime: 8/6/2014 9:06:15 PM
```

```
         ,macro_parms = /* %nrstr(data=sashelp.class,var=sex)  */
         ,hex16      = 1 /* convert numerics to hex16?         */
         ,semicolon  = 0 /* is each call (end of) a statement? */
         ,testing    = 0
)/ des = 'site: make macro statement then call'
   /*  ** store source /* */
;/* predecessor: callxinc, call execute a parameterized include
    successor  : calltext
RJF 7/18/2014 polishing for publication
*****/
%local column dsid list_parameters name n_obs n_vars
       rc  row  type time_start time_end;
%let semicolon = %eval(&semicolon
     or %lowcase(%scan(&macro_name ,1,%str( ))) eq put);
%let testing= %eval(   not(0 eq &testing)
                    or(    %sysfunc(getoption(mprint))
                                    eq %upcase(mprint)
                       and %sysfunc(getoption(source2))
                                    eq %upcase(source2) ));
%let time_start = %sysfunc(datetime(),hex16);

%**  description: assertions;
%**  purpose    : if fail then exit;
%if  not(%sysfunc(exist(&data))) %then %do;
     %put note: 0.1 &sysmacroname exit not exist(&data);
     %return;
     %end;
%let dsid  = %sysfunc(open (&data         ));
%let n_obs  = %sysfunc(attrn(&dsid,nobs ));
%let n_vars = %sysfunc(attrn(&dsid,nvars));
%if  not &n_obs or not &n_vars %then %do;
%put note: 0.2 &sysmacroname exit &data obs=&n_obs vars=&n_vars;
     %goto close_exit;
     %end;
%else
%put note: 1 &sysmacroname reading &data obs=&n_obs vars=&n_vars;

%** description: read data;
%** purpose    : make macro call, submit;
```

```
119     %*prepend macro_parms to list_parameters  note suffix=comma;
120     %if   %length(&macro_parms) %then
121           %let list_parameters = %unquote(&macro_parms),;
122           %else %let list_parameters = ;
123
124     %*** note:               fetchobs::read row;
125     %let rc       = %sysfunc(fetchobs(&dsid,&row ));
126     %do column = 1 %to &n_vars;
127         %let name = %sysfunc(varname (&dsid,&column));
128         %let num  = %sysfunc(varnum  (&dsid,&name  ));
129         %let type = %sysfunc(vartype (&dsid,&num   ));
130         %let name = %lowcase(&name);
131         %let type = %lowcase(&type);
132
133         %*** append string: parameter-n=;
134         %let list_parameters = &list_parameters&name=;
135         %*** append: for type=c: value, type=n: hex16(value);
136         %if   &type eq c
137            or(&type eq n and not &hex16) %then
138               %let list_parameters = &list_parameters%left(%sysfunc(
139                           getvar&type(&dsid,&num)));
140         %else %let list_parameters = &list_parameters%left(%sysfunc(
141            putn(%sysfunc(getvar&type(&dsid,&num)),hex16)));
142         %** append comma;
143         %if &column lt &n_vars %then
144             %let list_parameters = &list_parameters,;
145         %if &testing %then
146             %put note: 2 testing: &list_parameters;
147         %end;%* do column=;
148
149         %*** submit: note! no ending semicolon!;
150         %&macro_name(&list_parameters)
151         %** for testing: macro_name=put note  add semicolon;
152         %if &semicolon %then %do;
153             ;
154             %end;
155     %end;%*do row=;
156
157 %close_exit: %let rc = %sysfunc(close(&dsid));
158 %let time_end = %sysfunc(datetime(),hex16);
159 %put note: &sysmacroname used real time %sysfunc
160            (putn(&time_end.x-&time_start.x,time12.3));
161 %mend callmacr;
```

Summary

Conclusion

The macro function `callmacr.sas` can be used to read any list, a control data set, and produce a macro call from the values of variables in each observation. This routine hides the complexity of processing lists and removes that task from each list processing application. It solves the problem of passing numbers across step boundaries by converting both integers and reals to hexadecimal to maintain accuracy.

Suggested Reading

programs : Fehd [3] contains the programs in this paper.

predecessor : Fehd [6] SmryEachVar, a data-review routine, contains the parameterized include routine CallXinc, which is the prototype for this macro.

Fehd [8] List Processing with Call Execute describes the parameterized include routine CallXinc.

Fehd [9] contains more examples of the use of CallXinc.

successor : Fehd [4] Macro CallText uses the same algorithm as CallMacr to read a data set; it returns tokens within a statement constructed from the variable values in the observation.

Fehd [7] Database Vocabulary expands on the SmryEachVar suite and analyses the concept of cardinality ratio.

Fehd [12] contains a set of macros which expand on the SmryEachVar suite by using `callmacr.sas` for the list processing routine.

HIPO : IBM [14] HIPO — A Design Aid and Documentation Technique;
see also `http://en.wikipedia.org/wiki/HIPO`

lists : Fehd and Carpenter [13] List Processing Basics is the inspiration for dividing a list processing application program into two components, the processing subroutine and the routine that handles the macro loop which calls the subroutine.

Fehd [10] How to use SQL for list processing describes the various sql dictionaries that can be used to produce lists of data sets and variables.

Fehd [2] Making Lists contains programs to make lists of data sets, files, folders, and variable names.

logic : Fehd [11] Macro Design Ideas has truth tables for Boole's and De Morgan's logical operators.

parameters : Davis and Nelson [1] Passing Parameters to Tasks Launched by Servers and Schedulers

testing : Fehd [5] Writing Testing-Aware Programs

Continued on next page.

Bibliography

[1] Michael L. Davis and Gregory S. Barnes Nelson. No program is an island: Passing parameters to tasks launched by servers and schedulers. In *NorthEast SAS Users Group Annual Conference Proceedings*, 1999. URL http://www.nesug.org/Proceedings/nesug99/ad/ad080.pdf. 10 pp.; autoexec, theory: organization.

[2] Editor R.J. Fehd. Making lists. In *sasCommunity.org*, 2008. URL http://www.sascommunity.org/wiki/Making_Lists. how to make lists of data sets, and variable names using proc contents and sql; using data steps to get lists of files and folders.

[3] Editor R.J. Fehd. Macro Call-Macro. In *sasCommunity.org*, 2012. URL http://www.sascommunity.org/wiki/Macro_CallMacr. using SCL functions to read a data set and call macros.

[4] Editor R.J. Fehd. Macro Call-Text. In *sasCommunity.org*, 2012. URL http://www.sascommunity.org/wiki/Macro_CallText. using SCL functions to read a data set and return tokens within a statement.

[5] Ronald J. Fehd. Writing testing-aware programs that self-report when testing options are true. In *NorthEast SAS Users Group Annual Conference Proceedings*, 2007. URL http://www.nesug.org/Proceedings/nesug07/cc/cc12.pdf. Coders' Corner, 20 pp.; topics: options used while testing: echoauto, mprint, source2, verbose; variable testing in data step or macros; call execute; references.

[6] Ronald J. Fehd. SmryEachVar: A data-review routine for all data sets in a libref. In *SAS Global Forum Annual Conference Proceedings*, 2008. URL http://www2.sas.com/proceedings/forum2008/003-2008.pdf. Applications Development, 24 pp.; call execute, data review, data structure, dynamic programming, list processing, parameterized includes, utilities (writattr, writvalu) to repair missing elements in data structure; best contributed paper in ApDev.

[7] Ronald J. Fehd. Database vocabulary: Is your data set a dimension (lookup) table, a fact table or a report? In *Proceedings of the Western Users of SAS Software Annual Conference*, 2008. URL http://wuss.org/proceedings08/08WUSS%2520Proceedings/papers/dmw/dmw04.pdf. Databases and Warehouses, 8 pp.; cardinality ratio, composite key, database design, foreign key, grain, nlevels, normal forms, primary key, relational database, snapshots: accumulating or periodic.

[8] Ronald J. Fehd. List processing with call execute: Routine CallXinc for calling parameterized include programs using a data set as list of parameters. In *Pharmaceutical Industry SAS(R) Users Group Annual Conference Proceedings*, 2009. URL www.lexjansen.com/pharmasug/2009/ad/ad02.pdf. Applications Development, 10 pp.; call execute, data review, data structure, dynamic programming, list processing, parameterized includes, examples.

[9] Ronald J. Fehd. List processing routine CallXinc: Calling parameterized include programs using a data set as list of parameters. In *Proceedings of the Western Users of SAS Software Annual Conference*, 2009. URL www.lexjansen.com/wuss/2009/app/APP-Fehd2.pdf. Applications Development, 20 pp.; call execute, data review, data structure, dynamic programming, list processing, parameterized includes, examples.

[10] Ronald J. Fehd. How to use proc SQL select into for list processing. In *South Central SAS Users Group Annual Conference Proceedings*, 2010. URL http://analytics.ncsu.edu/sesug/2010/HOW06.Fehd.pdf. Hands On Workshop, 40 pp.; topics: writing constant text, and macro calls, using macro %do loops; references.

[11] Ronald J. Fehd. Macro design ideas: Theory, template, practice. In *SAS Global Forum Annual Conference Proceedings*, 2014. URL http://support.sas.com/resources/papers/proceedings14/1899-2014.pdf. 21 pp.; topics: logic, quality assurance, testing, style guide, documentation, bibliography.

[12] Ronald J. Fehd. Using cardinality ratio for fast data review. In *Proceedings of the Western Users of SAS Software Annual Conference*, 2014. Coders Corner, 14 pp.; successor of SmryEachVar, macros to calculate cardinality ratio and process discrete and continuous variables with procs freq, mode, and summary.

[13] Ronald J. Fehd and Art Carpenter. List processing basics: Creating and using lists of macro variables. In *SAS Global Forum Annual Conference Proceedings*, 2007. URL http://www2.sas.com/proceedings/forum2007/113-2007.pdf. Hands On Workshop, 20 pp.; comparison of methods: making and iterating macro arrays, scanning macro variable, writing calls to macro variable, write to file then include, call execute; using macro function nrstr in call execute argument; 11 examples, bibliography.

[14] IBM, editor. *HIPO — A Design Aid and Documentation Technique*. IBM Corporation, White Plains, NY, 1974. URL http://en.wikipedia.org/wiki/HIPO. Publication Number GC20-1851.

Continued on next page.

Closure

Contact Information: **Ronald J. Fehd** mailto:Ron.Fehd.macro.maven@gmail.com
http://www.sascommunity.org/wiki/Ronald_J._Fehd

About the author:

education:	B.S. Computer Science, U/Hawaii,	1986
	SAS User Group conference attendee	since 1989
	SAS-L reader	since 1994
experience:	programmer: 25+ years	
	data manager using SAS:	17+ years
	statistical software help desk:	7+ years
	author: 30+ SUG papers	
	sasCommunity.org: 300+ pages	
SAS-L:	author: 6,000+ messages to SAS-L since	1997
	Most Valuable SAS-L contributor:	2001, 2003

Trademarks