Paper 214-2023

List Processing Macro Call-Macro

Ronald J. Fehd, senior maverick, theoretical programmer,
Fragile-Free Software Institute

Abstract

description This paper provides an explanation of the general-purpose list processing routine Call-Macro in SAS® software. 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 Source Control Language (SCL) dataset functions that open a dataset, read and process each variable in an observation, and then close the data set.

audience intermediate macro users, applications programmers

keywords module, routine, subroutine, macro function sysfunc,
SCL dataset functions: attrn, exist, open, close, fetchobs, getvarc, getvarn, varname, vartype

In this paper:

List of programs:

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 dataset, in which each row is a set of values of the parameters of the subroutine. Finally, read the control dataset and generate calls to the macro containing the output subroutine.

Macro `callmacr.sas` is a list-processing routine which reads a control dataset, 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 dataset name
- `macro_name`, name of macro to call,
  the subroutine which produces output
- `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

---

overview

This introduction has the following topics.

- definition of a list
- program definitions
- developing subroutines
- making lists
- task: processing a list

---

definition of a list A list is defined in several different sets 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.

comp. science : A list is an example of an associative array. While the strictest definition defines a row as only a *key plus value,* in defining a dataset as an associative array the *key* may be one variable or a composite key: two or more variables. *Value* may be one or more labels of information which describe the *key*.

database : A database *fact table* contains three categories of columns: primary key, foreign key(s), fact(s). Foreign keys are a link to a *dimension table* which contains columns: primary key, and information about that key in one or more columns. A list corresponds to a dimension table.

SAS : In this article a dataset is used as a list. Some of the elements of a dataset 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, a subroutine.

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

! $\rightarrow$ A list is a control dataset; the variable names in the list match the names of the parameters of the list processing output subroutine.

---

program definitions

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

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

- **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.

- modules call modules, routines and subroutines
- routines call routines and subroutines
- subroutines do not call other programs; they produce output

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 dataset, 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 dataset
- style guide with naming conventions
- write testing-aware code

---

making lists

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

- `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 program 1 to provide a frequency procedure output dataset.

Program 1 call-macro-test-class.sas

```
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';
          title4  'key, label.1 label.2' ;
run;
```

task: processing lists

The task is to encapsulate and hide the complexity of reading a list, a control dataset, 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.

lst

```
list :: control dataset
key  label.1  label.2
value  COUNT  PERCENT
   F       9   47.3684
   M      10   52.6316
```

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

```
%*process;
%callmacr(data       = freq_class_sex
         ,macro_name = proc_this     )
%*tokens produced: ;
%proc_this(value=F,count= 9,percent=47.3684)
%proc_this(value=M,count=10,percent=52.6316)
```

These are the goals for the routine.

- process any dataset, 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 SAS Component Language (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 :
- `exist`, `open`, `close` of dataset
- `attrn` to fetch nobs and nvars of dataset
- `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

---

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 dataset 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)
```

loops

Two loops read the two-dimensional matrix of the dataset. 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
```

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 programs which are a minimum test of the macro.

- prototype
- macro print_subset
- testing program
- example module

prototype

Program 2 is the prototype program from which the print-subset macro was developed.

Program 2 example-sashelp-class-where.sas

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

macro print_subset

Program 3 is a macro subroutine used to print any subset of a dataset, given the libname, memname, variable (name), value, and information from the frequency procedure: count and percent.

Program 3 macro print_subset.sas

```
 /*    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;
```

testing program

Program 4 is the unit test of the macros `callmacr` and `print_subset`.

Program 4 call-macro-test-class.sas

```
* 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';
          title4  'key, label.1 label.2' ;
run;
%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 dataset

```
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 dataset
key   label.1   label.2
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)
```

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
```

example module

Program 5 is an example of a module which can be easily modified for any other dataset and variable.

Program 5 call-macro-demo-module.sas

```
 /*    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

Program 6 callmacr.sas

```
callmacr.sas documentation  page 1 of 4
/*       name: <UNC>\SAS-site\sas-macros\callmacr.sas
        author: Ronald J. Fehd  2023
                ----------------------------------------------------------
summary       : description: call macro using
                all values in data set row as parameters
      purpose : provide generic method to call 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: auto-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 =     work.freq_class_sex;
run;*necessary;
%callmacr(data       = work.freq_class_sex
         ,macro_name  = put note:
         ,macro_parms = %nrstr(data=sashelp.class,var=sex)
         )
log:
note:(data=sashelp.class,var=sex
     ,Sex=F,COUNT=4022000000000000,PERCENT=4047AF286BCA1AE7)
-------------------------------------------------------------------------
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_pe_C_Rcent;
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)
------------------------------------------------- */
```

```
%macro callmacr
        (data        = sashelp.class /* required  */
        ,macro_name  = put /* default for testing */
        ,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
        ,unquote     = 1 /* write notes to log? */
        )/ des = 'site: make macro statement then call'
        /*  ** store source /* */
        ;/* RJF 7/18/2014 polishing for publication
*** NOTE: _c_*: avoid name collisions w/data set vars ***/
%local _c_col _c_dsid _c_listparms _c_name _c_nobs _c_nvars
       _c_rc  _c_row  _c_type _c_semicolon
       _c_testing _c_time_start _c_time_end;
%let _c_semicolon = %eval(&semicolon
     or %lowcase(%scan(&macro_name ,1,%str( ))) eq put);
%let _c_testing= %eval( not(0 eq &testing)
     or(    %sysfunc(getoption(mprint))  eq %upcase(mprint)
        and %sysfunc(getoption(source2)) eq %upcase(source2)));
%let _c_time_start = %sysfunc(datetime(),hex16);
%let _c_hex16      = &hex16;
%let _c_unquote    = &unquote;

%**  description: assertions;
%**  purpose    : if fail then exit;
%if  not(%sysfunc(exist(&data))) %then %do;
     %put note: 0 &sysmacroname exiting: not exist(&data);
     %return;
     %end;
%let _c_dsid  = %sysfunc(open (&data         ));
%let _c_nobs  = %sysfunc(attrn(&_c_dsid,nobs ));
%let _c_nvars = %sysfunc(attrn(&_c_dsid,nvars));
%if  not &_c_nobs or not &_c_nvars %then %do;
     %put note: 1 &sysmacroname &data obs=&_c_nobs vars=&_c_nvars;
     %goto close_exit;
     %end;
%else
 %put note: 2 &sysmacroname reading &data obs=&_c_nobs vars=&_c_nvars;
```

```
%** description: read data;
%** purpose    : make macro call, submit;
%do _c_row = 1 %to &_c_nobs;
    %*prepare for macro_name  note suffix=comma;
    %if   %length(&macro_parms) %then
          %let _c_listparms = %unquote(&macro_parms),;
    %else %let _c_listparms = ;

    %*** note:                  fetchobs==read row;
    %let _c_rc       = %sysfunc(fetchobs(&_c_dsid,&_c_row ));
    %do _c_col = 1 %to &_c_nvars;
        %let _c_name = %sysfunc(varname (&_c_dsid,&_c_col ));
        %let _c_num  = %sysfunc(varnum  (&_c_dsid,&_c_name));
        %let _c_type = %sysfunc(vartype (&_c_dsid,&_c_num ));
        %let _c_name = %lowcase(&_c_name);
        %let _c_num  = %lowcase(&_c_num );
        %let _c_type = %lowcase(&_c_type);

        %*** append string: parameter-n=;
        %let _c_listparms = &_c_listparms&_c_name=;
        %*** append: for type=c: value, type=n: hex16(value);
        %if   &_c_type eq c
              or(&_c_type eq n and not &_c_hex16) %then
              %let _c_listparms = &_c_listparms%left(%sysfunc(
                   getvar&_c_type(&_c_dsid,&_c_num)));
        %else %let _c_listparms = &_c_listparms%left(%sysfunc(
              putn(%sysfunc(getvar&_c_type(&_c_dsid,&_c_num)),hex16)));
        %** append comma;
        %if &_c_col lt &_c_nvars %then
            %let _c_listparms = &_c_listparms,;
        %if &_c_testing %then
            %put note: 3 testing: &_c_listparms;
        %end;%* do columns;

        %*** submit: note! no ending semicolon!;
        %&macro_name(&_c_listparms)
        %** for testing: macro_name=put note  add semicolon;
        %if &_c_semicolon %then %do;
            ;
            %end;
    %end;%*do rows;

%close_exit: %let _c_rc = %sysfunc(close(&_c_dsid));
%let  _c_time_end = %sysfunc(datetime(),hex16);
%put  note: &sysmacroname used real time %sysfunc
      (putn(&_c_time_end.x-&_c_time_start.x,time12.3));
%mend callmacr;
```

Conclusion

The macro function `callmacr.sas` can be used to read any list, a control dataset, 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

predecessors : Fehd, "Writing Macro Do Loops with Dates from Then to When"

Fehd, "List Processing Routine CallXinc: Calling Parameterized Include Programs Using a Data Set as List of Parameters"

Fehd, "SmryEachVar: A Data-Review Routine for All Data Sets in a Libref" SmryEachVar, a data-review routine, contains the parameterized include routine CallXinc, which is the prototype for macro callmacr.

successors : Fehd, "Introduction To SCL Functions For Macro Programmers"

Fehd, "List Processing Macro Call-Text" Macro CallText uses the same algorithm as CallMacr to read a dataset; it returns tokens within a statement constructed from the variable values in the observation.

Fehd, "Using Cardinality Ratio for Fast Data Review" contains a set of macros which expand on the SmryEachVar suite by using `callmacr.sas` for the list processing routine.

HIPO : *IBM HIPO model — A Design Aid and Documentation Technique*

lists : Fehd and Carpenter, "List Processing Basics: Creating and Using Lists of Macro Variables" 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, "How To Use proc SQL select into for List Processing"

parameters : Davis and Nelson, "No Program Is An Island: Passing Parameters to Tasks Launched by Servers and Schedulers"

database : Kimball and Ross, *The Data Warehouse Toolkit, The Complete Guide to Dimensional Modeling, Second Edition*

Fehd, "Database Vocabulary: Is Your Data Set a Dimension (LookUp) Table, a Fact Table or a Report?"

testing : Fehd, "Writing Testing-Aware Programs that Self-Report when Testing Options are True" Writing Testing-Aware Programs

---

Author information

Ronald J. Fehd — Ron.Fehd.macro.maven at gmail dot com

LinkedIn — https://www.linkedin.com/in/ronald-fehd-5125991/

affiliation — senior maverick, theoretical programmer, Fragile-Free Software Institute

also known as — `macro maven` on SAS-L

References

Davis, Michael L. and Gregory S. Barnes Nelson (1999). "No Program Is An Island: Passing Parameters to Tasks Launched by Servers and Schedulers". In: *SouthEast SAS Users Group Conference Proceedings*. Systems Architecture, 10 pp. URL: https://analytics.ncsu.edu/sesug/1999/102.pdf.

Fehd, Ronald J. (2007). "Writing Testing-Aware Programs that Self-Report when Testing Options are True". In: *NorthEast SAS Users Group Conference Proceedings*. Coders' Corner, 20 pp.; topics: options used while testing: echoauto, mprint, source2, verbose; variable testing in data step or macros; call execute; references. URL: http://www.lexjansen.com/nesug/nesug07/cc/cc12.pdf.

— (2008a). "Database Vocabulary: Is Your Data Set a Dimension (LookUp) Table, a Fact Table or a Report?" In: *Western Users of SAS Software Annual Conference Proceedings*. 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. URL: http://wuss.org/proceedings08/08WUSS%2520Proceedings/papers/dmw/dmw04.pdf.

— (2008b). "SmryEachVar: A Data-Review Routine for All Data Sets in a Libref". In: *SAS Global Forum Annual Conference Proceedings*. 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. URL: http://www2.sas.com/proceedings/forum2008/003-2008.pdf.

— (2009). "List Processing Routine CallXinc: Calling Parameterized Include Programs Using a Data Set as List of Parameters". In: *Western Users of SAS Software Annual Conference Proceedings*. Applications Development, 20 pp.; call execute, data review, data structure, dynamic programming, list processing, parameterized includes, examples. URL: http://www.lexjansen.com/wuss/2009/app/APP-Fehd2.pdf.

— (2010). "How To Use proc SQL select into for List Processing". In: *SouthEast SAS Users Group Conference Proceedings*. Hands On Workshop, 40 pp.; writing constant text, and macro calls, using macro %do loops; URL: http://analytics.ncsu.edu/sesug/2010/HOW06.Fehd.pdf.

— (2013). "Writing Macro Do Loops with Dates from Then to When". In: *MidWest SAS Users Group Annual Conference Proceedings*. 20 pp.; topics: dates are integers, formats and functions to convert date references to integers, calculations, bibliography. URL: http://analytics.ncsu.edu/sesug/2013/CC-03.pdf.

— (2014). "Using Cardinality Ratio for Fast Data Review". In: *Western Users of SAS Software Annual Conference Proceedings*. Coders Corner, 14 pp.; successor of SmryEachVar, macros to calculate cardinality ratio and process discrete and continuous variables with procs freq, mode, and summary. URL: http://www.lexjansen.com/wuss/2014/98_Final_Paper_PDF.pdf.

— (2016). "List Processing Macro Call-Text". In: *MidWest SAS Users Group Annual Conference Proceedings*. Tools of Trade, 10 pp.; using %sysfunc with SCL functions to read a list, a control data set, and for each observation, return %unquoted text. URL: http://www.mwsug.org/proceedings/2016/TT/MWSUG-2016-TT06.pdf.

— (Oct. 2022). "Introduction To SCL Functions For Macro Programmers". In: *SouthEast SAS Users Group Conference Proceedings*. 20 pp.; URL: https://www.lexjansen.com/sesug/2022/SESUG2022_Paper_156_Final_PDF.pdf.

Fehd, Ronald J. and Art Carpenter (2009). "List Processing Basics: Creating and Using Lists of Macro Variables". In: *SouthEast SAS Users Group Conference Proceedings*. 8.of.9. URL: http://analytics.ncsu.edu/sesug/2009/HOW008.Fehd.Carpenter.pdf.

IBM HIPO model — A Design Aid and Documentation Technique (n.d.). URL: https://en.wikipedia.org/wiki/HIPO_model.

Kimball, Ralph and Margy Ross (2002). *The Data Warehouse Toolkit, The Complete Guide to Dimensional Modeling, Second Edition*. subtitle: Complete Guide to Dimensional Modeling; 17 chap., 387 pp., glossary: 29 pp., index: 18 pp. John Wiley & Sons, Inc., New York. URL: http://www.kimballgroup.com/html/booksDWT2.html.