Paper 097-2022

Introduction to scl for Macro Programmers

Ronald J. Fehd

Abstract

description: SAS® Component Language (scl), previously known as Screen Control Language, was developed for writing interactive applications using SAS software. It is the scripting language behind SAS/AF, SAS/FSP, and SAS/EIS software.
This paper examines the use of scl functions in writing macro definitions. The benefits include access to data attributes: memlabel, nobs and nvars, and variables attributes: varnum, varname, and vartype which can then be used to write allocations of macro variables for use with parameterized includes, as well as calling macros with named parameters.

purpose: The purpose of this paper is to compare examples of scl functions in a data step and with associated usage in macros.

audience: programmers writing macros

keywords: data: open, attrc, attrn, varname, varnum, vartype, close

directories: filename, dopen, dnum, dclose

In this paper:

List of programs:

Introduction

In this paper you will find comparisons of usage of scl functions, first in data step, then in macro.

Loops in SAS macro definitions depend on an upper bound; scl provides these via the `nobs` (n-rows) and `nvars` (n-columns) functions. After having

obtained these array dimensions it is a simple task to read each cell in a data set.

These are highlights from the programs below.

- print data with data set attributes
- write to file comma-separated values
- call execute a subroutine
- macro to call a macro

---

data review: sashelp.class

Program 1 shows a listing of and the data structure of SAS-supplied example data set `sashelp.class`.

This data set is used in the following sections.

Program 1 review-sashelp-class.sas

```
%let data = sashelp.class;
proc print  data  =        &data;
            title          &data;
proc sql;   describe table &data; quit;
```

---

print

```
sashelp.class
Obs Name     Sex  Age  Height  Weight
 1  Alfred   M     14    69.0   112.5
 2  Alice    F     13    56.5    84.0
 3  Barbara  F     13    65.3    98.0
...
17  Ronald   M     15    67.0   133.0
18  Thomas   M     11    57.5    85.0
19  William  M     15    66.5   112.0
```

---

describe

```
NOTE: SQL table SASHELP.CLASS was created like:
   create table SASHELP.CLASS( label='Student Data' )
  (Name   char(8),
   Sex    char(1),
   Age    num,
   Height num,
   Weight num
```

Data set information

overview

The first task for reading a data set is to fetch the values of upper bounds of the loops which read first rows, then columns within the row.
When writing macro definitions with these functions, it is important to determine in which scope — local, up-one-level, or global — the macro variables are allocated.
The list of topics in this section is:

- data attributes: label, n-obs, n-vars
- variable attributes: varnum, name, type

---

data attributes: label, n-obs, n-vars

Program 2 shows the use of the scl functions `open`, `attrc`, `attrn`, and `close` to list the values of data set attributes `label`, `nobs`, and `nvars`.

Program 2 data-attributes.sas

```
%let data = sashelp.class;
data _null_;
ds_id  = open (      "&data");
label  = attrc(ds_id,'label');
n_obs  = attrn(ds_id,'nobs' );
n_vars = attrn(ds_id,'nvars');
sys_rc = close(ds_id        );
put _all_;
stop;
run;
```

log

```
ds_id=1  label=Student Data  n_obs=19  n_vars=5  sys_rc=0
```

notes: The arguments to the scl functions are character strings: `"&data", 'label', 'nobs', 'nvars'`.

macro: data attributes

Program 3 shows the use of the function `%sysfunc` to access scl functions `open`, `attrc`, `attrn`, and `close` in a macro definition to list the values of data set attributes `label`, `nobs`, and `nvars`.
Note that this macro definition has neither `%global` nor `%local` statements for the macro variables so their default scope is local.

Program 3 data-attributes-macro.sas

```
%macro demo(data = sashelp.class);
%let ds_id  = %sysfunc(open (        &data ));
%let label  = %sysfunc(attrc(&ds_id, label ));
%let n_obs  = %sysfunc(attrn(&ds_id, nobs  ));
%let n_vars = %sysfunc(attrn(&ds_id, nvars ));
%let sys_rc = %sysfunc(close(&ds_id        ));
%put _local_;
%mend demo;
%demo;
```

notes: The values of the arguments to the scl functions do not need to be quoted: `&data, label, nobs, nvars`.

log

```
DEMO DATA   sashelp.class
DEMO DS_ID  1
DEMO LABEL  Student Data
DEMO N_OBS  19
DEMO N_VARS 5
DEMO SYS_RC 0
```

variable attributes: varnum, varname, vartype

Program 4 shows the use of the scl functions `open, attrn,` and `close` to fetch the values of variable attributes `varname` and `vartype`.

Program 4 data-loop-columns.sas

```
%let data = sashelp.class;
data _null_;
ds_id      = open   (      "&data");
n_vars     = attrn  (ds_id,'nvars');
put n_vars=;
do varnum  = 1 to n_vars;
   varname = varname(ds_id,varnum);
   vartype = vartype(ds_id,varnum);
   put varnum= varname= vartype=;
   end;
sys_rc     = close  (ds_id);
stop;
run;
```

log

```
n_vars=5
varnum=1 varname=Name   vartype=C
varnum=2 varname=Sex    vartype=C
varnum=3 varname=Age    vartype=N
varnum=4 varname=Height vartype=N
varnum=5 varname=Weight vartype=N
```

notes: The values of vartype are in (C,N).

macro: variable attributes

Program 5 shows the use of the function `%sysfunc` to access the scl functions `open, attrn, varname, vartype` and `close` in a loop in a macro definition which fetches the values of variable attributes `varname` and `vartype` for each variable.

Program 5 data-loop-columns-macro.sas

```
*name: data-loop-columns-macro.sas;
%macro demo(data = sashelp.class);
%let ds_id       = %sysfunc(open   (       &data  ));
%let n_vars      = %sysfunc(attrn  (&ds_id,nvars  ));
%put &=n_vars;
%do varnum  = 1 %to &n_vars;
    %let varname = %sysfunc(varname(&ds_id,&varnum));
    %let vartype = %sysfunc(vartype(&ds_id,&varnum));
    %put &=varnum &=varname &=vartype;
    %end;
%let sys_rc      = %sysfunc(close  (&ds_id        ));
%mend demo;
%demo;
```

log

```
N_VARS=5
VARNUM=1 VARNAME=Name   VARTYPE=C
VARNUM=2 VARNAME=Sex    VARTYPE=C
VARNUM=3 VARNAME=Age    VARTYPE=N
```

```
VARNUM=4 VARNAME=Height VARTYPE=N
VARNUM=5 VARNAME=Weight VARTYPE=N
```

User-written functions and scope

overview

When writing macro definitions which are functions, i.e., they return values, it is important to understand how SAS software handles the allocation of macro variables within the appropriate scope. It seems obvious that there are two scopes: local or global; there is yet a third: up-one-level.
This section examines the use of the `%global` and `%local` statements in macro definitions and shows methods of controlling scope in macro definitions.
The list of topics in this section is:

- calling functions to return values as side effects
- functions to return data or variable attributes

calling functions to return values as side effects

overview

SAS software uses the prefix `call` in various function names to indicate the the results occur or happen elsewhere.
In this section we examine two macro definitions which use the `%global` statement in a program and within a macro definition.
This is the list of topics in this section:

- call_atributes
- call_atributes_global

call_attributes

Program 6 shows the use of the the function `%sysfunc` to access the scl functions `open, attrn,` and `close` to fetch the values of data-set attributes `label, nobs,` and `nvars` in a macro definition;
note that the names of desired attributes must be allocated before calling the macro.

Program 6 macro call_attrib.sas

```
 /*name: sas-macros\call_attrib.sas
macro function returns data attributes
in macro variable(s) previously allocated
usage:
%global _label  _n_obs  _n_vars;
%call_attrib(libref.memname);
%put _user_;
****** */
%macro call_attrib(libref_memname);
%let  ds_id  = %sysfunc(open (&libref_memname));
%let _label  = %sysfunc(attrc(&ds_id,label   ));
%let _n_obs  = %sysfunc(attrn(&ds_id,nobs    ));
%let _n_vars = %sysfunc(attrn(&ds_id,nvars   ));
%let  sys_rc = %sysfunc(close(&ds_id         ));
%mend call_attrib;
```

call_attributes_global

Program 7 shows the use of the function %sysfunc to access the scl functions open, attrc, attrn, and close in a macro definition to fetch the values of data-set attributes label, nobs, and nvars and return them to global macro variables provided in the parameter named list.

Program 7 macro call_attrib_global.sas

```
 /*name: sas-macros\call_attrib_global.sas
macro function returns data attributes
in global macro variable(s) in parameter: list
usage:
%call_attrib_global(libref.memname,_label);
%put _label;
****** */
%macro call_attrib_global(libref_memname,list) ;
%if  %length(&list) %then       %global &list  ;
%let  ds_id  = %sysfunc(open (&libref_memname));
%let _label  = %sysfunc(attrc(&ds_id,label   ));
%let _n_obs  = %sysfunc(attrn(&ds_id,nobs    ));
%let _n_vars = %sysfunc(attrn(&ds_id,nvars   ));
%let  sys_rc = %sysfunc(close(&ds_id         ));
%mend call_attrib_global;
```

---

commentary

The above macro definitions show how to control the scope of macro variables. Their constraints are knowing in advance the naming conventions of the macro variables returned by the functions. In this case each name had a prefix of underscore: (_).
The next section shows how to write in-line functions where the program determines the naming convention.

---

functions to return data or variable attributes

overview

This section shows macro definitions written as in-line functions: they return values within a macro variable assignment statement.
It includes review of sql dictionaries from which the naming conventions are derived.
This is the list of topics in this section:

- sql dictionary.tables
- macro memlabel
- macro memlabel-test
- macro nobs
- macro nvars
- demo in-line functions
- sql dictionary.columns
- macro vartype

---

sql dictionary.tables

Program 8 shows the use of the sql allocation of macro variables.
Note that dictionary.tables.nvar is singular.

Program 8 sql-dictionary-tables.sas

```
*name: sql-dictionary-tables.sas;
%let libname = sashelp;
```

```
%let memname = class;
proc sql noprint;
         describe table      dictionary.tables;
         select   memlabel,  nobs,  nvar
           into  :memlabel, :nobs, :nvars
           from             dictionary.tables
          where   libname eq "%upcase(&libname)"
            and   memname eq "%upcase(&memname)"
            and   memtype eq "%upcase(data)";
         %put echo &=memlabel &=nobs &=nvars;
         quit;
```

describe

```
NOTE: SQL table DICTIONARY.TABLES was created like:
  (libname  char(8)   label='Library Name',
   memname  char(32)  label='Member Name',
   memtype  char(8)   label='Member Type',
   memlabel char(256) label='Data Set Label',
   nobs     num       label='Number of Physical Observations',
   nvar     num       label='Number of Variables',
```

log

```
80  %put echo &=memlabel &=nobs &=nvars;
echo MEMLABEL=Student Data   NOBS=19   NVARS=5
```

macro memlabel

Program 9 shows an in-line macro function which returns the label of a data set.

Program 9 macro memlabel.sas

```
 /*name: sas-macros\memlabel.sas
macro function returns data-set label:
libref.memname(label="memlabel")
usage:
title3     "%memlabel(libref.memname)";
%let mvar = %memlabel(sashelp.class);
notes:
* naming convention: sql dictionary.libnames: memlabel
* caveat: labels may contain special characters!
          ampersand, percent, semicolon
****** */
%macro memlabel(libref_memname);
%let   ds_id  = %sysfunc(open (&libref_memname));
%***   return ; %sysfunc(attrc(&ds_id,label   ))
%let   sys_rc = %sysfunc(close(&ds_id         ));
%mend  memlabel;
```

notes: `%* return ;` This statement is a macro comment which is a reminder that the function returns its value in-line, not, as in previous programs `call_attrib` and `call_attrib_global`, as side effects.
`%sysfunc(attrc(&ds_id,label))` Note that the function call does not have a closing semicolon.
See program 12, demo-inline-functions.sas, for an example of usage.

macro nobs

Program 10 shows an in-line macro function which returns the number-of-observations of a data set.

Program 10 macro nobs.sas

```
 /*name: sas-macros\nobs.sas
macro function returns nobs(data-set)
%let mvar = %nobs(sashelp.class);
****** */
%macro nobs(libref_memname);
%let   ds_id  = %sysfunc(open (&libref_memname));
%***   return ; %sysfunc(attrn(&ds_id,nobs    ))
%let   sys_rc = %sysfunc(close(&ds_id         ));
%mend  nobs;
```

macro nvars

Program 11 shows an in-line macro function which returns the number-of-variables of a data set.

Program 11 macro nvars.sas

```
 /*name: sas-macros\nvars.sas
macro function returns nvars(data-set)
%let mvar = %nvars(sashelp.class);
****** */
%macro nvars(libref_memname);
%let   ds_id  = %sysfunc(open (&libref_memname));
%***   return ; %sysfunc(attrn(&ds_id,nvars   ))
%let   sys_rc = %sysfunc(close(&ds_id         ));
%mend  nvars;
```

demo of in-line functions

Program 12 shows usage of in-line macro functions `memlabel, nobs` and `nvars`.

Program 12 demo-inline-functions.sas

```
proc print data = &data;
           title1  "data= &data "
                   "label= %memlabel(&data) ";
           title2  "nobs= %nobs    (&data) "
                   "nvars= %nvars   (&data) ";
run;
```

listing

```
data=sashelp.class label=Student Data
nobs = 19 nvars = 5
Obs   Name      Sex   Age   Height   Weight
1     Alfred    M      14    69.0    112.5
2     Alice     F      13    56.5     84.0
3     Barbara   F      13    65.3     98.0
```

sql dictionary.columns

Program 13 shows the use of the sql allocation of macro variables to fetch the varnum and type of a variable.

Program 13 sql-dictionary-columns.sas

```
*name: sql-dictionary-columns.sas;
%let libname = sashelp;
%let memname = class;
%let name    = sex;
```

```
proc sql noprint;
         describe table    dictionary.columns;
         select   varnum,  type
           into  :varnum, :type
           from           dictionary.columns
          where   libname      eq "%upcase(&libname)"
            and   memname      eq "%upcase(&memname)"
            and   memtype      eq "%upcase(data)"
            and   upcase(name) eq "%upcase(&name)";
         %put echo &=varnum &=name &=type;
         quit;
```

describe

```
NOTE: SQL table DICTIONARY.COLUMNS was created like:
  (libname  char(8)   label='Library Name',
   memname  char(32)  label='Member Name',
   memtype  char(8)   label='Member Type',
   name     char(32)  label='Column Name',
   type     char(4)   label='Column Type',
   length   num       label='Column Length',
   varnum   num       label='Column Number in Table',
   label    char(256) label='Column Label',
   format   char(49)  label='Column Format',
   informat char(49)  label='Column Informat',
```

log

```
89  %put echo &=varnum &=name &=type;
echo VARNUM=2 NAME=sex  TYPE=char
```

notes: The values of sql dictionary.columns.type are in (char,num);

macro vartype

Program 14 shows the use of the scl functions `open, attrn,` and `close` to fetch the values of variable attribute `vartype`.
Note that the parameter, `name`, is used to look up the `varnum`, which value is then used to look up `vartype`.

Program 14 macro vartype.sas

```
 /*name: sas-macros\vartype.sas
macro function returns C | N
usage:
%let mvar = %vartype(x,y);
%put &=mvar;
%if %vartype(libref.memname,name) eq C %then ...;
****** */
%macro vartype(data,name);
%let   ds_id  = %sysfunc(open   (       &data  ));
%let   varnum = %sysfunc(varnum (&ds_id,&name  ));
%***   return ; %sysfunc(vartype(&ds_id,&varnum))
%let   sys_rc = %sysfunc(close  (&ds_id        ));
%mend  vartype;
```

demo vartype

Program 15 shows the use of the in-line macro function `vartype` in a macro definition.

Program 15 demo-vartype.sas

```
options obs=3;
%macro subset(data,name,value);
proc print data  = &data
          (where = (&name eq
%if %vartype(&data,&name) eq C %then "&value";
%else                                 &value ;
         ));
         title1 "&data";
         title2 "&name eq &value";
run;
%mend  subset;
%subset(sashelp.class,sex,F);
%subset(sashelp.class,age,11);
```

log

```
84         %subset(sashelp.class,sex,F);
MPRINT(SUBSET):   proc print data = sashelp.class (
MPRINT(SUBSET):  where = (sex eq "F" ));
MPRINT(SUBSET):   title1 "sashelp.class";
MPRINT(SUBSET):   title2 "sex eq F";
MPRINT(SUBSET):   run;
NOTE: There were 3 observations read from the data set SASHELP.CLASS.
      WHERE sex='F';
```

listing

```
sashelp.class
sex eq F
Obs  Name      Sex  Age  Height  Weight
2    Alice     F     13    56.5    84.0
3    Barbara   F     13    65.3    98.0
4    Carol     F     14    62.8   102.5
```

List processing: calling subroutines from data

overview

This section provides subroutine data-loop-write-csv and programs and macro definitions which are routines designed to call subroutines.
This is the list of topics in this section:

- data: write csv
- reading rows and columns
- macro: data-loop-rows-columns
- frequency of variable sex
- utility include: put-user
- calling includes
- utility macro: put_parmbuff
- calling macros

data: write csv

Program 16 shows the use of the scl functions `open, attrn,` and `close` to fetch the values of variable attributes `name, type` and `value` for each row in the data set.
Note that the loop `do varnum = 1 to n_cols` writes the header record: list of variable names.

Program 16 data-loop-write-csv.sas

```
%let data = sashelp.class;
data _null_;
     file log; *file "&filename..csv";
ds_id         = open      (      "&data");
n_rows        = attrn     (ds_id,'nobs' );
n_cols        = attrn     (ds_id,'nvars');
do varnum     = 1 to n_cols;
   varname    = varname   (ds_id,varnum );
   put                           varname @;
   if varnum lt n_cols then put ','     @;
   end;
put;
do row        = 1 to n_rows;
   rc         = fetchobs  (ds_id,row   ) ;
   do varnum  = 1 to n_cols;
      varname = varname   (ds_id,varnum) ;
      vartype = vartype   (ds_id,varnum) ;
      if vartype eq 'C' then do;
         value_c = getvarc(ds_id,varnum) ;
         put value_c                    @;
         end;
      else do;
         value_n = getvarn(ds_id,varnum) ;
         put value_n                    @;
         end;
      if varnum lt n_cols then put ','  @;
      end;
   put ;%*'...'/;*newline;
   end;
sys_rc  = close(ds_id);
stop;
run;
```

output

```
Name    ,Sex ,Age ,Height ,Weight
Alfred  ,M   ,14  ,69     ,112.5
Alice   ,F   ,13  ,56.5   ,84
...
Thomas  ,M   ,11  ,57.5   ,85
William ,M   ,15  ,66.5   ,112
```

notes: Spaces have been inserted into this output for readability.

reading rows and columns

Program 17 shows the use of the scl functions `open, attrn,` and `close` to fetch the values of variable attributes `name, type` and `value` for each row in the data set.

Program 17 data-loop-rows-cols.sas

```
%let data = sashelp.class;
data _null_;
ds_id            = open    (      "&data");
n_rows           = attrn   (ds_id,'nobs' );
n_cols           = attrn   (ds_id,'nvars');
put n_rows= n_cols=;
do row           = 1 to n_rows;
   rc            = fetchobs(ds_id,row    );
   do varnum     = 1 to n_cols;
      varname    = varname (ds_id,varnum );
      vartype    = vartype (ds_id,varnum );
      if vartype eq 'C' then do;
         value_c = getvarc (ds_id,varnum );
         put varname= value_c=;
         end;
      else do;
         value_n = getvarn (ds_id,varnum );
         put varname= value_n=;
         end;
      end;
      put '...'/;*newline;
   end;
sys_rc  = close(ds_id);
stop;
run;
```

log

```
n_rows=19 n_cols=5
varname=Name   value_c=Alfred
varname=Sex    value_c=M
varname=Age    value_n=14
varname=Height value_n=69
varname=Weight value_n=112.5
...
varname=Name   value_c=William
varname=Sex    value_c=M
varname=Age    value_n=15
varname=Height value_n=66.5
varname=Weight value_n=112
```

macro: data-loop-rows-cols

Program 18 shows the use of function %sysfunc to access the scl functions open, attrn, and close in a macro definition to fetch the values of variable attributes name, type and value for each row in the data set.
Note that functions getvarc and getvarn have been replaced with a reference to macro variable type: getvar&type.

Program 18 data-loop-rows-cols-macro.sas

```
%macro demo(data = sashelp.class);
%let ds_id        = %sysfunc(open       (       &data  ));
%let n_rows       = %sysfunc(attrn      (&ds_id,nobs   ));
%let n_cols       = %sysfunc(attrn      (&ds_id,nvars  ));
%put &=n_rows &=n_cols;
%do row           = 1 %to &n_rows;
    %let rc       = %sysfunc(fetchobs   (&ds_id,&row   ));
    %do varnum    = 1 %to &n_cols;
        %let name = %sysfunc(varname    (&ds_id,&varnum));
        %let type = %sysfunc(vartype    (&ds_id,&varnum));
        %let value= %sysfunc(getvar&type(&ds_id,&varnum));
        %put &=name &=value;
        %end;
    %put ...;%*newline;
    %end;
%let sys_rc = %sysfunc(close(&ds_id));
%mend demo;
%demo;
```

log

```
N_ROWS=19 N_COLS=5
NAME=Name   VALUE=Alfred
NAME=Sex    VALUE=M
NAME=Age    VALUE=14
NAME=Height VALUE=69
NAME=Weight VALUE=112.5
...
NAME=Name   VALUE=William
NAME=Sex    VALUE=M
NAME=Age    VALUE=15
NAME=Height VALUE=66.5
NAME=Weight VALUE=112
```

frequency of variable sex

Program 19 provides a list of the values of a variable.
This data-set is used in the following programs.

Program 19 make-list-values-freq.sas

```
%let data = sashelp.class;
%let name = sex;
proc freq  data   = &data;
           tables   &name  / noprint
              out = list_values;
proc print data =         &syslast;
proc sql;  describe table &syslast; quit;
```

print

```
Obs  Sex  COUNT   PERCENT
1    F        9   47.3684
2    M       10   52.6316
```

describe

```
NOTE: SQL table WORK.LIST_VALUES was created like:
   create table WORK.LIST_VALUES
                  (label='Frequency Counts and Percentages')
  (Sex     char(1),
   COUNT   num     label='Frequency Count',
   PERCENT num     label='Percent of Total Frequency'
```

utility: put-user

Program 20 is a utility used for testing in program 21, call-include.sas.

Program 20 put-user.sas

```
%*name:    sas-includes\put-user.sas
  compare to sas-macros\put_parmbuff.sas;
%put _user_;
```

calling includes

Program 21 is a list-processing routine: it reads a data set and, for each row, writes a call to allocate a macro variable for each variable and value, then writes the call to the subroutine.

Program 21 call-include.sas

```
%include project(make-list-values-freq);
%let data       = &syslast;
%let subroutine = site_inc(put-user);
data _null_;
     attrib name  length = $ 32
            type  length = $  1
            stmnt length = $256;
ds_id       = open    (      "&data");
n_rows      = attrn   (ds_id,'nobs' );
n_cols      = attrn   (ds_id,'nvars');
do row      = 1 to n_rows;
   rc       = fetchobs(ds_id,row    );
   do col   = 1 to n_cols;
      name  = varname (ds_id,col    );
      type  = vartype (ds_id,col    );
      stmnt = catx(' ','%let',name,'=');
      if type eq 'C' then
           stmnt = catt(stmnt,getvarc(ds_id,col));
      else stmnt = catt(stmnt,getvarn(ds_id,col));
      call execute(cats('%nrstr(',stmnt,';)'));
      end;
   stmnt = "%include &subroutine";
   call    execute(cats('%nrstr(',stmnt,';)'));
   end;
sys_rc  = close(ds_id);
stop;
run;
run;
```

log

```
NOTE: CALL EXECUTE generated line.
1     + %let Sex =F;
2     + %let COUNT =9;
3     + %let PERCENT =47.368421053;
4     + %include site_inc(put-user);

5     + %let Sex =M;
6     + %let COUNT =10;
7     + %let PERCENT =52.631578947;
8     + %include site_inc(put-user);
109    run;
```

utility: put_parmbuff

Program 22 is a utility macro used to echo the parameter list of any macro subroutine called by program 23, macro call_macro.sas.

Program 22 macro put_parmbuff.sas

```
 /*name: sas-macros\put_parmbuff.sas;
utility for testing list-processing macros
which call subroutines
usage: %callmacr(data=freq,subroutine=put_parmbuff)
compare to sas-includes\put-user
****** */
%macro put_parmbuff/parmbuff;
%put   echo: &=syspbuff;
%mend  put_parmbuff;
```

calling macros

Program 23 is a routine, a program which calls subroutines, in this case macros.
Its default subroutine name is `put echo` which resolves to simply
`%put echo (<parameter-list>)`.
This makes testing the subroutine straightforward.
Next test is to use program 22, macro put_parmbuff.sas.

Program 23 macro call_macro.sas

```
%macro   call_macro
        (data       = &syslast
        ,subroutine = put echo);
%let ds_id      = %sysfunc(open       (&data       ));
%let n_rows     = %sysfunc(attrn      (&ds_id,nobs ));
%let n_cols     = %sysfunc(attrn      (&ds_id,nvars));
%do row         = 1 %to    &n_rows;
   %let rc      = %sysfunc(fetchobs   (&ds_id,&row ));
   %let parms   = ;
   %do col      = 1 %to    &n_cols;
      %let name = %sysfunc(varname    (&ds_id,&col ));
      %let type = %sysfunc(vartype    (&ds_id,&col ));
      %let parms
     =&parms&name=%sysfunc(getvar&type(&ds_id,&col ));
      %if &col lt &n_cols %then  %let parms = &parms,;
```

```
      %end;
   %&subroutine(&parms);
   %end;
%let sys_rc     = %sysfunc(close      (&ds_id));
%mend call_macro;
```

macro call_macro, test

Program 23, call_macro-test.sas, is a non-trivial list-processing module; its default parameter for testing is `put echo` which returns a complete statement: `%put echo(<...>);`.
Program 24 shows the testing sequence:

1. default: put echo
2. subroutine=put_parmbuff 22 macro put_parmbuff.sas
3. subroutine=demo

Program 24 call_macro-test.sas

```
*name: call_macro-test.sas;
%include project(make-list-values-freq);
options nomlogic nosymbolgen source2;
%*include project(call_macro);
%*include site_mac(put_parmbuff);
%call_macro()
%call_macro(subroutine=put_parmbuff)
%macro demo(sex=.,count=.,percent=.);
%put  _local_;
%mend  demo;
%call_macro(subroutine=demo)
```

log

```
 92         %call_macro()
 echo(Sex=F,COUNT=9,PERCENT=47.3684210526315)
 echo(Sex=M,COUNT=10,PERCENT=52.6315789473684)

 93         %call_macro(subroutine=put_parmbuff)
 echo: SYSPBUFF=(Sex=F,COUNT=9,PERCENT=47.3684210526315)
 MPRINT(CALL_MACRO):  ;
 echo: SYSPBUFF=(Sex=M,COUNT=10,PERCENT=52.6315789473684)
 MPRINT(CALL_MACRO):  ;

94         %macro demo(sex=.,count=.,percent=.);
95         %put  _local_;
96         %mend  demo;
97         %call_macro(subroutine=demo)
DEMO COUNT 9
DEMO PERCENT 47.3684210526315
DEMO SEX F
MPRINT(CALL_MACRO):  ;
```

Files in directory

Program 25 provides a list of the files in a directory.
It uses the `filename` function to assign a fileref `my_dir` which is then used in the directory-open function: `dopen`;
the number-of-files function: `dnum`, fetches the upper bound of the loop reading the list of files.

the `dread` function fetches the name of the file.

Program 25 files-in-directory.sas

```
%let directory = .;
%let select    = 1;   *all files;
%let select    = scan(filename,-1,'.') eq 'sas';
data list_filenames (keep   = directory filename);
     attrib directory length = $256
            filename  length = $ 72;
     retain directory "&directory";
rc      = filename("my_dir","&directory");
dir_id  = dopen   ("my_dir");
n_files = dnum    (dir_id);
do i = 1 to n_files;
   filename = dread(dir_id,i);
   if &select then output;
   end;
rc = dclose(dir_id);
run;
proc print data = &syslast;
run;
```

log

```
Obs  directory  filename
  1  ./sas      autoexec.sas
  2  ./sas      data-loop-columns.sas
  3  ./sas      data-loop-columns-macro.sas
  4  ./sas      data-loop-rows-cols.sas
  5  ./sas      data-loop-rows-cols-macro.sas
  6  ./sas      data-loop-write-csv.sas
```

macro: files in directory

Program 26 is the macro of program 25, files-in-directory.sas.

Program 26 macro files_in_directory.sas

```
%macro files_in_directory
      (directory  = .
      ,select     = %nrstr(&ext eq txt)
      ,subroutine = put echo);
%let rc           = %sysfunc(filename(my_dir,&directory));
%let dir_id       = %sysfunc(dopen   (my_dir           ));
%let n_files      = %sysfunc(dnum    (&dir_id          ));
%do i = 1 %to &n_files;
    %let filename = %sysfunc(dread   (&dir_id,&i       ));
    %let ext      = %scan&filename,-1,.);
    %if %unquote(&select) %then %do;
        %&subroutine(directory=&directory
                    ,filename=&filename);
        %end;
    %end;
%let rc           = %sysfunc(dclose  (&dir_id          ));
%mend files_in_directory;
```

log

```
88  %files_in_directory
89  (directory = /home/ron.fehd.macro.m/sas-scl-intro
```

```
90  ,ext       = log)
echo(directory=./sas,filename=data-loop-columns.log)
echo(directory=./sas,filename=data-loop-columns-macro.log)
echo(directory=./sas,filename=data-loop-rows-cols.log)
echo(directory=./sas,filename=data-loop-rows-cols-macro.log)
echo(directory=./sas,filename=sql-dictionary-tables-columns.log)
```

Suggested reading

predecessors	Fehd, “Advanced Programming Concepts: History of the List Processing and Cardinality Ratio Memes”
	Fehd, “Applications Design and Development, a case study of the EDA Summarize-Each-Variable Suite”
	Fehd, “Do we need Macros? An Essay on the Theory of Application Development”
	Fehd, “Using the SmryEachVar Data Review Suite”
sql for research	Fehd, “How To Use proc SQL select into for List Processing”
conditions	Fehd, “True is not False: Evaluating Logical Expressions”
	Fehd, “Macro Code to Test Existence of Various Objects”
call-text	Fehd, “List Processing Macro Call-Text”
call-macro	Fehd, “List Processing Macro Call-Macro”
call-dates	Fehd, “Writing Macro Do Loops with Dates from Then to When”
call-include	Fehd, “List Processing Routine CallXinc: Calling Parameterized Include Programs Using a Data Set as List of Parameters”
%global/readonly	Langston, “New Macro Features Added in SAS 9.3 and SAS 9.4”
mopen	Langston, “A Macro to Verify a Macro Exists”

Conclusion

SCL functions enable macro programmers to open data sets and read data attributes: nobs, and nvars. Loops can be constructed with these upper bounds to write csv files, call execute an include, or call a macro. Scope of macro variables allocated in macro definitions can be managed with in-line functions. Other functions enable reading of list of files in a directory and calling subroutines to process each file.

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

Fehd, Ronald J. (2009a). "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.

— (2009b). "Using the SmryEachVar Data Review Suite". 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-Fehd1.pdf.

— (2010). "How To Use proc SQL select into for List Processing". In: *SouthEast SAS Users Group Conference Proceedings*. Hands On Workshop, 40 pp.; topics: writing constant text, and macro calls, using macro %do loops; references. 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). "List Processing Macro Call-Macro". In: *MidWest SAS Users Group Annual Conference Proceedings*. Coders Corner, 19 pp.; using %sysfunc with SCL functions to read a list, a control data set, and for each observation, call a macro with variable names and values as named parameters. URL: http://www.mwsug.org/proceedings/2014/BB/MWSUG-2014-BB04.pdf.

— (2015a). "Applications Design and Development, a case study of the EDA Summarize-Each-Variable Suite". In: *SouthEast SAS Users Group Conference Proceedings*. Hands On Workshops, 22 pp. URL: http://www.lexjansen.com/sesug/2015/96_Final_PDF.pdf.

— (2015b). "Do we need Macros? An Essay on the Theory of Application Development". In: *MidWest SAS Users Group Annual Conference Proceedings*. Beyond Basics, 10 pp. URL: http://www.lexjansen.com/mwsug/2015/BB/MWSUG-2015-BB-11.pdf.

— (2016a). "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.

— (2016b). "Macro Code to Test Existence of Various Objects". In: *SouthEast SAS Users Group Conference Proceedings*. Coders Corner, 9 pp.; macro functions %sysfunc and ifc; data: exist, catalog: cexist, filename: fileexist, fileref of file: fexist, fileref of folder: fileref, libref: libref. URL: https://analytics.ncsu.edu/sesug/2016/CC-187_Final_PDF.pdf.

— (2016c). "True is not False: Evaluating Logical Expressions". In: *SouthEast SAS Users Group Conference Proceedings*. Beyond the Basics, 9 pp.; Boolean algegra, Boolean logic, De Morgan's law, evaluation, logical operators, sql joins. URL: https://analytics.ncsu.edu/sesug/2016/BB-186_Final_PDF.pdf.

— (2018). "Advanced Programming Concepts: History of the List Processing and Cardinality Ratio Memes". In: *Western Users of SAS Software Annual Conference Proceedings*. URL: http://www.lexjansen.com/wuss/2018/71_Final_Paper_PDF.pdf.

Langston, Rick (2013). "A Macro to Verify a Macro Exists". In: *SAS Global Forum Annual Conference Proceedings*. 5 pp.; dopen, dclose, fclose, mopen. URL: https://support.sas.com/resources/papers/proceedings13/339-2013.pdf.

— (2015). "New Macro Features Added in SAS 9.3 and SAS 9.4". In: *SAS Global Forum Annual Conference Proceedings*. 6 pp.; %put &=mvar, %global/readonly, %symexecdepth, %symexecname, %sysmacexist, %sysmacdelete, %sysmstoreclear, sys* automatic macro variables. URL: https://support.sas.com/resources/papers/proceedings15/SAS1575-2015.pdf.