

Advanced Programming Concepts: History of the List Processing and Cardinality Ratio Memes

Ronald J. Fehd, Stakana Analytics

Abstract	Description: A statement in a natural language contains three parts: subject, verb and object. A statement in a computer language contains only two parts: verb and an object; the subject, or actor, is the computer's operating system. The predecessor of the meme <i>list processing</i> is the computer language LISP, in which every statement is a function call and the object is a list. The list processing paradigm of programming contains these steps: 1. identify an object, an <i>item</i>; 2. write a function, process, or procedure for an item; 3. prepare a <i>list</i> of items; 4. use a loop on the list, to process each item. This paper reviews the author's development of the concept of list processing and its implementation in SAS® software. Purpose: The purpose of this exposition is to highlight the author's papers published on these topics and to provide a critique of earlier ideas. This is accomplished by a review of the development of processes for calculating the <i>cardinality ratios cardinality types</i> of the variables in a data set. Audience: programmers Keywords: arrays of macro variables; cardinality ratio; compiled or executed contents, extended; database vocabulary; data structure and algorithm; dosubl function; list processing; macro arrays; scl functions: open, close, attrn(nobs, nvars), fetchobs, getvarc, getvarn, varname, vartype, style: This paper uses font changes to identify SAS words in text, documentation words in <i>italics</i> and notable key words in SMALL CAPS; e.g.: the libname statement allocates a <i>libref</i>; CARDINALITY TYPES are in FEW, MANY and UNIQUE. quote: If you think of standardization as the best that you know today, but which is to be improved tomorrow — you get somewhere. — Henry Ford
-----------------	--

In this paper

Introduction	2
Definition of a list	2
Learning a computer language	4
Arrays of macro variables	4
Database vocabulary	7
The three basic lists	7
Variable names	7
Data set names: memnames	9
Values	10

Loops: processes for lists	12
Specific Routines	12
Call execute an %include	14
Demo macro function which reads a data set	15
Call-macro	16
Call-text	17
A function to calculate cardinality ratios and types	18
N steps	19
Two steps with freq.nlevels and scl functions	20
One step with scl functions and dosubl	23
Cardinality types of all memnames in a libref	25
Summary	26
References	27

Introduction

Overview

This is the overview, which consists of a list of topics in this section.

- definition of a list
 - learning a computer language
 - arrays of macro variables
 - database vocabulary
-

Definition of a list

Overview

This section contains these topics.

- list processing definitions
 - contents listing
 - contents program
 - contents output data set
-

List processing definitions

list : contains items
item : can be an attribute, or a list
attribute : description, information, number, or string
CARDINALITY : of a list is the number of items in the list
same as dimension of an array, number of observations in data set
SAS list : is a CONTROL DATA SET, where each row contains list of values for a process, and the variable names are parameter names of that process; it is like a database DIMENSION TABLE, see page 7;
Notes: compare to proc format options cntlin, cntlout (control-in and -out) and data structure provided by proc contents

contents listing

```
proc contents data = sashelp.class;

The CONTENTS Procedure
Data Set Name  SASHELP.CLASS      Observations      19
Member Type    DATA              Variables          5
Label          Student Data

Alphabetic List of Variables and Attributes
#  Variable    Type    Len
-  -
3  Age         Num     8
4  Height      Num     8
1  Name        Char    8
2  Sex         Char    1
5  Weight      Num     8
```

Notes: ATTRIBUTES of a data set include: *libref* (sashelp) is a reference to a folder; data set name (class) is the name of a file in the folder; data set label (Student Data) provides a description of the data set; dimensions of the data matrix are height (Observations 19) and width (Variables 5); list of the attributes of variables in the data set includes the ROW IDENTIFIERS, variable number (labeled #) and name and type in (Char,Num).

contents program

```
*name: make-list-names-contents-demo.sas;
%let data = sashelp.class;
PROC contents data = &data noprint
    out = list_names
        (keep = libname memname varnum nob
            name type length format label);
run; * updates sysnobs and syslast;
PROC sql; describe table &syslast;
quit;
proc print data = &syslast;
run;
```

contents output data set

```
create table WORK.LIST_NAMES
  LIBNAME char(8) label='Library Name',
  MEMNAME char(32) label='Library Member Name',
  NAME char(32) label='Variable Name',
  TYPE num label='Variable Type', <---<<< num
  LENGTH num label='Variable Length',
  VARNUM num label='Variable Number',
  LABEL char(256) label='Variable Label',
  FORMAT char(32) label='Variable Format',
  NOBS num label='Observations in Data Set'
```

Notes: type is numeric which differs from listing of type in (1=Num,2=Char)

contents output data set

Obs	LIBNAME	MEMNAME	NAME	TYPE	LENGTH	VARNUM	LABEL	FORMAT	NOBS
1	SASHELP	CLASS	Age	1	8	3			19
2	SASHELP	CLASS	Height	1	8	4			19
3	SASHELP	CLASS	Name	2	8	1			19
4	SASHELP	CLASS	Sex	2	1	2			19
5	SASHELP	CLASS	Weight	1	8	5			19

Notes: Type in in (1=Num,2=Char), see standarization on page 7

Learning a computer language

- variables local or global type (boolean, char, num)
- conditions if then... else add code, or branch
- loops conditional exit while, leave, until
enumerate (start, stop, step)
itemize
- functions macro, method, process, procedure
- setup for startup configuration, autoexec, libraries
- compile or execute? data structure or algorithm?

Notes: Q: What is the difference between *process* and *procedure*?

process : is a set of numbered steps which are always performed;
testing is simple because output is consistent

procedure : is a set of choices which may add extra statements
— e.g., options — or branch:
if type eq char then ... else if type eq num then ...;
testing is complex due to multiple paths through the code

- citations:**
- learning a computer language Statz [49], Heilmann and W3C [45]
 - conditions: using logical operators *and*, *or*, *not* [42]
 - sysfunc and ifc: [28] writing testing-aware programs: [22]
 - loops [23]
 - macro design [36]
 - setup for startup: autoexec companion, [37],
and sysparm companion, [40]

Arrays of macro variables

Overview

This section contains these topics.

- data array and loops
- warning when only single type
- subroutine: make list values sort
- make macro array: data, symputx
- log: array of global macro variables
- make macro array: sql
- demo macro %do loop
- programming issues
- citations

data array and loops

Here is a generic method to write each row of a data set to the log.

```
*name: demo-data-array.sas;
%let lib_data = sashelp.class;
*let lib_data = sashelp.class(keep=age height weight);*type=n;
DATA _null_;
    if 0 then set &lib_data;          *read data structure;
    array _c(*) $32 _character_;      *fragile;
    array _n(*)      _numeric_ ;      *fragile;
do _obs = 1      to      _n_obs;      *loop, enumerated;
    set &lib_data nobs = _n_obs
              point = _obs;
    do _i = 1 to dim(_c);
        putlog _c(_i)= @;
    end;
    do _i = 1 to dim(_n);
        putlog _n(_i)= @;
    end;
    put;*CR/LF: newline;
end;
stop;
run;
```

Notes: The array statement is paired with the dim function.
The *dimension* of an array is the CARDINALITY of the array.
Compare to programs cx-include page 14
and demo-macro-function page 15.

warning when only single type

The above program gives a warning when all variables are a single type.

```
2 %let lib_data = sashelp.class(keep=age height weight);*type=n;
3 DATA _null_;
4     if 0 then set &lib_data;
5     array _c(*) $32 _character_;      *fragile;
WARNING: Defining an array with zero elements.
```

Notes: See how to eliminate this warning in program cx-include on page 14.

subroutine: make list values sort

This program is used as a subroutine in the next two examples.

```
*name: make-list-values-sort.sas;
PROC sort data = &lib_data (keep = &var)
    nodupkey
    out = list_values;
    by  &var;
run;
%put echo &=sysnobs;
```

make macro array: data, symputx

This is a classic program which uses the row number, `_n_`, as a *natural key*.

```
*name: demo-make-macro-array-data.sas;
%let lib_data = sashelp.class;
%let var      = sex;
%include 'make-list-values-sort.sas';
DATA _null_;
    set &syslast end = endofile nobs = dimension;
    call symputx(catt('value',_n_),&var);
    if endofile then call symputx("dim_values",dimension);
run;
%put _global_;
```

log: array of global macro variables

This log shows the macro variables of the parameters `lib_data` and `var`, and the sequentially-numbered macro variables with prefix `value` and the `CARDINALITY` of the macro array `dim_values`.

```
GLOBAL LIB_DATA    sashelp.class
GLOBAL VAR         sex
GLOBAL VALUE1      F
GLOBAL VALUE2      M
GLOBAL DIM_VALUES  2
```

make macro array: sql

Arrays of macro variable can also be produced with the `sql` procedure.

```
*name: demo-make-macro-array-sql.sas;
%let lib_data = sashelp.class;
%let var      = sex;
%include 'make-list-values-sort.sas';
PROC sql noprint;
    select &var
    into   :value1 -
    from   &syslast;
quit;
%let dim_values = &sqllobs;
%put _global_;
run;
```

Notes: the high value may be also coded into `:value1 - :value&sysmaxlong`

demo macro %do loop

Arrays of macro variables are referenced in a macro `%do` loop using the double ampersand syntax.

```
*name: demo-echo-macro-array.sas;
%include 'demo-make-macro-array-data.sas';
%include 'demo-make-macro-array-sql.sas';

%macro echo_items();
%do i = 1 %to &dim_values;
    %put value&i: &&value&i;
    *code for each item;
%end;
%mend echo_items;
%echo_items()
```

programming issues

The name of the macro array is hard-coded and is separate from the macro `%do` loop.

There is no function for the dimension, it must be fetched.

Macro variables are created in the global symbol table.

Macro with `%do` loop reads the global symbol table.

Code inside loop cannot be tested as a unit.

All these issues are addressed in the section on loops, page 12.

- citations:**
- macro array using `call symput` used in [12] and [13] [14]
 - macro array using `proc sql select into` [16]
 - macro array as 1. function which returns dimension, and 2. procedure so that calling macro can declare macro variables local [1]
 - macro `do_loop` [7]
-

Database vocabulary

database contains columns	primary key : row identifier, UNIQUE	NATURAL KEY has values 1–nobs
	foreign keys : categories, FEW, link to primary key of dimension table	
	composite key : set of foreign keys	
	fact : continuous, MANY, measurement or quantity; summable	
	text : information about primary key	
	boolean : event success	
	date : and/or time: event granularity for start/end of periodic snapshot	
database contains tables	fact : is a record of events	ex: transaction history, inventory observation
	contains primary key, foreign keys, facts	
	dimension : is also known as a lookup table	
	contains primary key, text	
	snapshots : are called reports	
	periodic : e.g. monthly (grain) summary	
	accumulating : tasks of project, budget record with monthly sums	
citations:	database vocabulary, [32] and Kimball and Ross [46]	

The three basic lists

Lists	In programming we use the lists of variables (names), data sets (memnames), and values. These lists are produced with these processes.	
	names :	contents, data with scl functions, sql dictionary.columns
	memnames :	contents, sql dictionary.tables
	values :	frequency, sort, sql, summary
citations:	• Other commonly-used lists are available from sql dictionaries; these include user- and system-defined global macro variables, <i>catrefs</i>, <i>filerefs</i>, <i>librefs</i>, options, and running text in footnotes and titles. How to use sql select into for list processing [29] code: [4] • SAS software maintains other functions in catalogs; these include user-defined formats and macro definitions. SASautos companion, [18] lists macro catalogs, code: [3]	

Variable names

Overview	Procedures contents and sql produce a list of variable names; scl functions can be used in a data step. Each method differs in the values of type. contents (1,2) scl (C,N) sql (char,num) The programs shown below standardize type into (c,n). This section contains these topics. • list-names, data structure • list-names, printed • list names, contents • list names, scl • list names, sql	
-----------------	---	--

list-names, data structure

```
create table work.list_names
  libname char(8),
  memname char(32),
  varnum num,
  name char(32),
  type char(1),
  length num,
  format char(40),
  label char(40)
```

list-names, printed

libname	memname	varnum	name	type	length	format	label
sashelp	class	1	Name	c	8		
sashelp	class	2	Sex	c	1		
sashelp	class	3	Age	n	8		
sashelp	class	4	Height	n	8		
sashelp	class	5	Weight	n	8		

list-names, contents

```
*name: make-list-names-contents-x.sas;
%put echo &libname..&memname;
PROC contents data = &libname..&memname noprint
  out = list_names
  (keep = libname memname varnum name nobs
    type length format label );
run;
DATA &syslast (label = "&memname");
  if 0 then do; * arrange data structure ;
    set &syslast(keep = libname memname nobs varnum name);
    attrib type length = $1;
    set &syslast(keep = length format);
    set &syslast(keep = label);
  end;
  retain _max_length_char 0;
  drop _; * _temp vars;
do until(endofile);
  set &syslast (rename = (type = _type_n))
    end = endofile;
  if _type_n = 1 then type = 'n';
  else do; type = 'c';
    _max_length_char = max(_max_length_char,length);
  end;
  output;
end;
call symputx('_max_length_char',_max_length_char);
run;
%put echo &_max_length_char;
PROC sort data = &syslast; *add attribute: sorted-by;
  by name;
run;
%put trace make-list-names-contents-x ending;
```

list-names, scl

```
*name: make-list-names-scl-x.sas;
%put echo &libname..&memname;
DATA list_names(label = "&memname");
  attrib libname length = $ 8
    memname length = $32
    n_obs length = 8
    varnum length = 8
    name length = $32
    type length = $ 1
    length length = 8
    format length = $40
    label length = $40;
  drop _; ** temp vars;
  retain libname "&libname" memname "&memname"
    _max_length_char n_obs 0;

_dsid = open("&libname..&memname");
```

```

_n_vars = attrn(_dsid,'nvar');
n_obs   = attrn(_dsid,'nobs');
do varnum = 1 to _n_vars;
    name      = varname  (_dsid,varnum) ;
    type      = lowercase(vartype  (_dsid,varnum));
    length    = varlength(_dsid,varnum) ;
    format    = varfmt   (_dsid,varnum) ;
    label     = varlabel (_dsid,varnum) ;
    output;
    if type eq 'c' then
        _max_length_char = max(_max_length_char,length);
    end;
_rc = close(_dsid);
call symputx('_max_length_char',_max_length_char);
run;
%put echo &=_max_length_char;
%put trace make-list-names-scl-x ending;

```

list-names, sql

```

*name: make-list-names-sql-x.sas      note: does not have nobs;
%put echo &libname..&memname;
PROC sql noprint;
    create table list_names (label = "&memname") as
    select libname, memname, varnum, name, type length=1,
           length, format, label
    from dictionary.columns           %*list-names;
    where libname eq "%upcase(&libname)"
           and memname eq "%upcase(&memname)"
           and memtype eq 'DATA';
    select max(length) into :_max_length_char   trimmed
    from   &syslast   where type eq 'c';
    quit;
%put echo &=_max_length_char;
%put trace make-list-names-sql-x ending;

```

citations: Code for the above programs and others is available on [2].

Data set names: memnames

Overview

This list is called *memnames* because that is the variable name in the output data sets of both the `contents` and `sql` procedures.

- list-memnames, data structure
- list-memnames, printed
- list-memnames, contents
- list-memnames, sql
- programming issues

The programs shown below add `memnum` with values (1 : *n_obs*), which is the NATURAL KEY described on page 7.

list-memnames, data structure

```

create table WORK.LIST_MEMNAMES
  libname char(8)   label='Library Name',
  memnum   num,      label='Member Number',
  memname  char(32)  label='Member Name',
  nobs     num       label='Number of Physical Observations',
  nvars    num       label='Number of Variables',
  memlabel char(256) label='Data Set Label'

```

list-memnames, printed

```
libname memnum memname nobs nvars memlabel
SASHELP 1 AACOMP 1544 4
SASHELP 2 AARFM 61 4
SASHELP 3 ADSMSG 426 6
SASHELP 4 AFMSG 1090 6
SASHELP 5 AIR 144 2 airline data (monthly: JAN49-DEC60)
...
```

list-memnames, contents

```
*name: make-list-memnames-contents.sas;
%put echo &=libname;
PROC contents data = &libname._all_ noprint
    out = list_memnames
        (keep = libname memname nobs memlabel);
run;
%put echo &=sysnobs;
DATA &syslast (label = "&libname");
    attrib libname length=$ 8 label='library name'
           memnum length=$ 8 label='member number'
           memname length=$ 32 label='member name'
           nobs length=$ 8 label='number of observations'
           nvars length=$ 8 label='number of variables'
           memlabel length=$256 label='member label';
    retain libname "&libname" memnum nvars 0;
set &syslast;
nvars +1;
by memname;
if last.memname then do;
    memnum +1;
    output;
    nvars = 0;
end;
run;
%put trace make-list-memnames-contents ending;
```

list-memnames, sql

```
*name: make-list-memnames-sql.sas;
PROC sql noprint;
    create table list_memnames(label = "&libname") as
    select libname,
           monotonic() as memnum label='member number',
           memname, nobs, nvar as nvars, memlabel
    from dictionary.tables
    where libname eq "%upcase(&libname)"
           and memtype eq 'DATA';
    quit;
%put trace make-list-memnames-sql ending;
```

programming issues

The sql dictionaries of all *librefs* are refreshed before each procedure call; the contents procedure reads only the *libref* specified and is therefore faster.

This information is also available in a set of *sashelp* views; these views read from sql dictionaries and are deprecated by the author for that reason.

Values Overview

This section shows the macro `proc-freq` and the data structure. The purpose of this macro is produce a frequency of each variable, standardize the data structure and stack the outputs into one data set.

This is the list of topics in this section.

- macro `proc-freq`
- list-values, data structure
- list-values, printed

macro proc-freq

This macro is called by two programs, demo-call-macro-call-execute on page 12 and demo-call-macro-sql-basic on page 13. Compare to macro proc-freq-dosub on page 23.

```
%macro proc_freq(name,type);
%put echo &libname..&memname..&name &=type;
%let type = %lowercase(%substr(&type,1,1));
PROC freq data = &libname..&memname;          *fragile: global;
    tables &name / noprint
        out = frequency
            (rename = (&name = valu_&type));
run;
DATA &syslast;
    attrib memname length = $32
           name length = $32
           valu_c length = $32          /*fragile;
/**      valu_n length = $&_max_length_char  /**robust*/
           count length = 8
           percent length = 8
           level length = 8;
    retain memname "&memname" name "&name"
           valu_c ' ' valu_n . ;
set &syslast;
level = _n_;                                *natural key;
run;
PROC append data = &syslast
    base = list_values;
run;
%put trace &sysmacroname(&=name) ending;
%mend proc_freq;
```

citations: This macro is adapted from [20].

list-values, data structure

```
create table WORK.LIST_VALUES
libname char(8),
memname char(32),
varnum num,
name char(32),
valu_c char(??),      <---<<<
valu_n num,
count num,
percent num,
leven num
```

list-values, printed

This report shows cardinality of each variable on its last line <---<<<.

Obs	memname	name	valu_c	valu_n	count	percent	level
1	class	Age		11.0	2	10.5263	1
2	class	Age		12.0	5	26.3158	2
[snipped]							
5	class	Age		15.0	4	21.0526	5
6	class	Age		16.0	1	5.2632	6 <---<<<
7	class	Height		51.3	1	5.2632	1
8	class	Height		56.3	1	5.2632	2
[snipped]							
22	class	Height		69.0	1	5.2632	16
23	class	Height		72.0	1	5.2632	17 <---<<<
24	class	Name	Alfred	.	1	5.2632	1
25	class	Name	Alice	.	1	5.2632	2
[snipped]							
41	class	Name	Thomas	.	1	5.2632	18
42	class	Name	William	.	1	5.2632	19 <---<<<
43	class	Sex	F	.	9	47.3684	1
44	class	Sex	M	.	10	52.6316	2 <---<<<
45	class	Weight		50.5	1	5.2632	1
[snipped]							
59	class	Weight		150.0	1	5.2632	15 <---<<<

programming issues

The length of variable `valu_c` is marked as *fragile*; note that the programs that produce the data set `list-names` shown above on 8 – 9 also provide a macro variable `max-length-char`.

Loops: processes for lists

Overview

This section contains the specific routines that are the predecessors to the three general-purpose, reusable routines.

- specific solutions
 - type macro calls, how to automate this?
 - call execute macro
 - sql select into
- general solutions, reusable
 - cx-include: call execute an %include
 - call-macro: call a macro
 - call-text: create and expand references to macro variable in text

Notes: Both call-macro and call-text are macro functions; they do not return SAS statements, only macro language text.

type the macro calls

In the beginning, cut and paste the list of variable names from the contents output and type in the constant text of the macro calls.

```
*name: demo-typed-list-values.sas;
/** use list from proc contents printed output;
Alphabetic List of Variables and Attributes
#   Variable   Type   Len
3   Age        Num    8
4   Height     Num    8
1   Name       Char    8
2   Sex        Char    1
5   Weight     Num    8
/*****/

%let libname = sashelp;
%let memname = class;

%proc_freq(Age ,Num )
%proc_freq(Height,Num )
%proc_freq(Name ,Char)
%proc_freq(Sex ,Char)
%proc_freq(Weight,Num )
PROC print;
run;
```

Specific Routines

call execute macro calls

The call execute routine has been around since SAS.v6. This program shows how to call a macro with two parameters.

```
*name: demo-call-macro-call-exec.sas;
%let libname = sashelp;
%let memname = class;
%include 'make-list-names-contents-x.sas';
DATA _null_;
do until(endofile);
    set list_names end = endofile;
    *call exec of a macro is fragile without nrstr;
    call execute(catt('%proc_freq(name=',name,',type=',type,')'));
end;
stop;
run;
```

Notes: This solution may fail, not because of the code in this program, but when the macro definition contains what I call complexity: it contains any of these statements: %do, %if, call symput, or symputx.

The error is difficult to diagnose because the macro will pass a unit test and this program will work with simple macros but not with complex ones.

An explanation of the timing issues is in Fehd and Carpenter [44].

The solution with %nrstr is shown in program cx-include on page 14.

sql select into

The sql dictionary.columns eliminates the separate step of making the list of variable names.

```
*name: demo-call-macro-sql-basic.sas;
%let libname = sashelp;
%let memname = class;
PROC sql noprint;
    select catt('%proc_freq('      /*fragile length= 200;
               ,name,',',type,')')
    into :list                                /*fragile length=65534;
        separated by ' '
    from dictionary.columns                /*list-names;
    where libname eq "%upcase(&libname)"
        and memname eq "%upcase(&memname)"
        and memtype eq 'DATA';
    quit;
&list
%syndel list;
run;
PROC print;
run;
```

Notes: Ah, yes, this is my most spectacular fail! I published this in the fall of 2007 [21], and one week later got an e-mail from a user complaining that it broke on a data set with three thousand (3,000) variables!

It may fail for either of two reasons.

- cat function : the default length of text returned by any of the cat functions is \$200; the short length results in truncation of the macro call, which has no closing parenthesis. This can be fixed by adding a length clause after the function call: catt(...) length = \$nnnn.
- macro variable : the maximum length of text in a macro variable is $(2^{16}) - 2 = 65,534$; a possible solution is to reduce the name of the macro definition to a single character: %macro z(...) otherwise, this error cannot be fixed; consider one of the robust solutions, shown next.
- citations:** A tutorial on sql is available in [29].

Call execute an %include

Overview

This is the main algorithm of `CallXproc.sas` from the `SmryEachVar` suite of `sgf.2008` [24]. This program was published later as `CallXinc.sas` [27].

Compare to the demonstration program `demo-data-array.sas` on page 5.

Note that the length of the variable `_stmt` is marked as fragile. The value is `max(for i=1 to n-vars (length(%let var(i)=value(i);)) ,length(%include &cx_include;))`

where `var(i)` is `length(name(i))` and `value(i)` is `length(i)`.

citations: Robust code for `cx-include` is available as `callxinc` on [26] which has a data step to calculate the maximum length for `_stmt`.

cx-include

```
*name: cx-include.sas;
%put echo &=cx_data &=cx_include;
DATA _null_;
  if 0 then set &cx_data;
  attrib _stmtnt length = $128          /*fragile;
         _name   length = $ 32
         _i      length = 4;
  array _var_c(*) _character_ ;          *robust: _name;
  array _var_n(*) _numeric_ ;           *robust: _i;
do until(endofile);
  set &cx_data end = endofile;
  do _i = 1 to dim(_var_c) -2;           *note: dim()-2;
    _name = vname(_var_c(_i));
    _stmtnt = catx(' ', '%let', _name, '=', _var_c(_i));
    link cx_stmtnt;
  end;
  do _i = 1 to dim(_var_n) -1;           *note: dim()-1;
    _name = vname(_var_n(_i));
    _stmtnt = catx(' ', '%let', _name, '=', _var_n(_i));
    link cx_stmtnt;
  end;
  _stmtnt = "%include &cx_include";
  link cx_stmtnt;
end;
stop;
cx_stmtnt: call execute(cats('%nrstr(', _stmtnt, ')'));
return;
run;
```

Notes: This program hides the complexity of the `call execute(...%nrstr())` in the data step subroutine labeled `cx_stmtnt` which is called with the `link` statement.

Example usage of this program is shown in
`make-list-cr-types-n-steps` on page 19 and
`make-list-cr-types-1-step-libname` on page 25.

Demo macro function which reads a data set

Overview

This demonstration macro shows the common code in macros `call_macro` and `call_text`.

Compare to programs `demo-data-array.sas` on page 5. and `cx-include` on page 14.

This program uses `scl` functions like `make-list-names-scl-x` on page 8.

Notes: This functions calls either of the functions `getvarc` or `getvarn` with the reference `getvar&type`. This trick eliminates the conditional statements `%if &type eq c %then getvarc(); %else getvarn()`.

demo macro function

```
%MACRO demo_macro_function
    (data = sashelp.class);
    %local dsid n_obs n_vars obs rc name type value;
%let dsid = %sysfunc(open (&data ));
%let n_obs = %sysfunc(attrn(&dsid,nobs ));
%let n_vars = %sysfunc(attrn(&dsid,nvars));
%do obs = 1 %to &n_obs;
    %let rc = %sysfunc( fetchobs(&dsid,&obs));
    %do varnum = 1 %to &n_vars;
        %let name = %sysfunc( varname(&dsid,&varnum));
        %let type = %sysfunc( vartype(&dsid,&varnum));
        %let value = %sysfunc(getvar&type(&dsid,&varnum));
        %put echo &name = &value;
    %end;
%put;
%end;
%let rc = %sysfunc(close(&dsid));
%mend demo_macro_function;
```

log

```
echo Name = Alfred
echo Sex = M
echo Age = 14
echo Height = 69
echo Weight = 112.5

echo Name = Alice
echo Sex = F
echo Age = 13
echo Height = 56.5
echo Weight = 84
```

Call-macro

Overview

The purpose of macro `call_macro` is to assemble a macro call of the form:
`%macro-name(var(i)=value(column(i),row(j)),...)`

The `%do` loops for `obs` and `varnum` and the fetch of `name` and `type` are exactly the same as shown above in `demo-macro-function`.

call-macro

```
%MACRO call_macro
    (data      = sashelp.class /* required */
     ,macro_name = put);      /* default for testing */
%let dsid    = %sysfunc(open (&data ));
%let n_obs   = %sysfunc(attrn(&dsid,nobs ));
%let n_vars  = %sysfunc(attrn(&dsid,nvars));
%do obs = 1 %to &n_obs;
    %let list_parameters =;
    %let rc               = %sysfunc(fetchobs(&dsid,&obs));
    %do varnum = 1 %to &n_vars;
        %let name    = %sysfunc(varname (&dsid,&varnum));
        %let type    = %sysfunc(vartype (&dsid,&varnum));
        /* add name=value to list;
        %let list_parameters=&list_parameters&name=%left(
            %sysfunc(getvar&type(&dsid,&varnum)));
        /* add comma to end of parameter list;
        %if &varnum lt &n_vars %then
            %let list_parameters = &list_parameters,;
        %end;
        %&macro_name(&list_parameters);/*semicolon not required!;
    %end;
%let rc = %sysfunc(close(&dsid));
%mend call_macro;
```

citations: Macro `callmacr` is described in [34], code is available on [5].

Call-text

Overview

The purpose of macro `call_text` is to assemble macro assignment statements and then expand `&text` which contains references to `&var1--&varn`.
`%let var1=value1;...;%let varn=valuen;%unquote(&text)`

call-text

```
%MACRO call_text
    (data   = sashelp.class
     ,text   = %nrstr(%put echo: &=name sex=&sex;)
     ,global = 0); /* make mvars global? */
%let _global = %eval(not(0 eq &global));
%let _text    = &text;
%let _dsid    = %sysfunc(open (&data      ));
%let _n_obs   = %sysfunc(attrn(&_dsid,nobs ));
%let _n_vars  = %sysfunc(attrn(&_dsid,nvars));
%do _obs = 1 %to &_n_obs;
    %let _rc      = %sysfunc(fetchobs   (&_dsid,&_obs  ));
    %do _varnum = 1 %to &_n_vars;
        %let _name = %sysfunc(varname   (&_dsid,&_varnum));
        %let _type = %sysfunc(vartype   (&_dsid,&_varnum));
        %if &_global %then %global &_name;
        %else          %local  &_name;
        %let &_name= %sysfunc(getvar&_type(&_dsid,&_varnum));
        %*** note functions:  getvarc and getvarn;
    %end;
    %unquote(&_text)
%end;
%let _rc = %sysfunc(close(&_dsid));
%mend call_text;
```

programming issues

Macros `call-macro` and `call-text` fail when text variables such as labels contain special characters of the macro language, ampersand (&), percent (%), or function delimiter, comma (,), or statement delimiter, semicolon (;), or unmatched (single or double) quotes or parentheses.

The solution is to drop those variables, `data=x(drop=label)`.

citations: Macro `calltext` is described in [41], code is available on [6].

Code for macros `call-macro` and `call-text` contains tests for the existence of the data set; tests for existence of various SAS objects is shown in [43] code: [9]

Macro `dateloop` is a function which returns a series of dates and surrounding text; it is described in [31] code: [8]

A function to calculate cardinality ratios and types

Overview

The cardinality of a set is the number of elements in the set. The cardinality of a data set is the number of rows in the data set. The cardinality of a variable is the number of distinct values within the variable; in SAS this is called *n-levels* and is available from the frequency procedure. The CARDINALITY RATIO (CR) of a variable is the cardinality divided by the data set nobs. The CARDINALITY TYPE (cr-type) is determined by comparing the CR with the mean of all the CR in the data set.

This section contains these programs.

- N steps page 19
- Two steps with freq.nlevels and scl functions page 20
- One step with scl and dosubl page 23
- Cardinality ratio types of all memnames in a libref page 25

This overview contains these topics.

- cr-types, data structure
- cr-types, printed
- algorithm

cr-types, data structure

```
create table WORK.CR_TYPES(label='class nobs=19 nvars=5')
  memname   char(32),
  varnum    num,
  cr_type   char(10),
  card_ratio num format=BESTD7.5,
  n_levels  num format=BEST8. label='Number of Levels',
  name      char(32)          label='Table Variable',
  type      char(1),
  length    num,
  format    char(49),
  label     char(256)
```

cardinality types, printed

memname	varnum	cr_type	card_ n_ ratio	levels	name	type	length	format	label
class	1	.unique		1	19 Name	c	8		
class	2	few	0.10526	2	2 Sex	c	1		
class	3	few	0.31579	6	6 Age	n	8		
class	4	many	0.89474	17	17 Height	n	8		
class	5	many	0.78947	15	15 Weight	n	8		

algorithm

1. fetch n-obs: nobs(data set) or from list-memnames
2. for each variable, fetch n-levels
3. cardinality-ratio = n-levels / n-obs
4. calculate mean of cardinality ratios
5. cardinality-ratio-type = compare cardinality ratio to mean

citations: The first paper to identify cardinality ratio was [25]; other papers include [32] and [33]. Macro `cr-calc`, shown in [35], is the basis for the n-steps programs shown next. This paper also includes programs for the mode and summary procedures.

N steps

Overview

Implementation of this algorithm requires $7 + n$ -vars steps.
This is the list of topics in this section.

- make-list-cr-types-n-steps
 - proc-freq-out-name
 - make-list-card-ratios-from-work-name
 - proc-smry-cr
 - make-list-types-from-cr
-

make-list-cr-types-n-steps

This program may also use the macro `calltext` for the loop.

```
*name: make-list-cr-types-n-steps.sas;
%let libname = sashelp;
%let memname = class;
%include 'make-list-names-contents-x.sas';          *steps=2;
/**** loop: freq of each variable *****/
%let cx_data = &syslast;
%let cx_include = 'proc-freq-out-name.sas';        *steps=nvars;
%include 'cx-include.sas';                        *steps=1;
/*****
*call_text(data = &syslast
            ,text = %nrstr(%include 'proc-freq-out-name.sas';)
            ,global = 1)/*****/
%include 'make-list-card-ratios-from-work-name.sas'; *steps=1;
%include 'proc-smry-cr.sas';                       *steps=1;
%include 'make-list-types-from-cr.sas';             *steps=1;
PROC print;                                         *steps=1;
            title3 "Cardinality Ratios and Types: &libname..&memname";
run;
```

proc-freq-out-name.sas

```
*name: proc-freq-out-name.sas;
PROC freq data = &libname..&memname;
      tables &name / noprint
            out = freq_&name;          * fragile: length=$32;
run;
```

Compare to programs `proc-freq` page 11 and `proc-freq-dosub` page 23.

make-list-card-ratios-from-work-name.sas

```
*name: make-list-card-ratios-from-work-name.sas;
PROC contents data = work._all_          noprint
      out = list_n_levels
      (keep = memname name nobs label
       where = ( memname like 'FREQ_%'
                  and label eq ' ' ))
      rename =(nobs = n_levels));
run;
PROC sort data = &syslast;
      by name;
DATA list_card_ratios;
do until(endofile);
      merge list_names &syslast (keep = name n_levels)
            end = endofile;
      by name;
      card_ratio = n_levels / nobs;
      output;
end;
stop;
run;
```

proc-smry-cr.sas

```
*name: proc-smry-cr.sas;
PROC summary data      = list_card_ratios;
      output out = _mean_cr (drop = _freq_ _type_)
      mean(card_ratio)= _mean_cr;
run;
```

make-list-types-from-cr.sas

```
*name: make-list-types-from-cr.sas;
DATA list_cr_types;
      drop _:;
set _mean_cr;
putlog 'echo' _mean_cr=;

do until(endofile);
      set list_card_ratios      end = endofile;
      select;
            when(n_levels      eq      1) cr_type = 'n-levels=1';
            when(card_ratio eq      1) cr_type = '.unique';
            when(card_ratio gt _mean_cr) cr_type = 'many';
            otherwise              cr_type = 'few';
      end;
      output;
      end;
stop;
run;
```

programming issues

This algorithm contains many steps which are interdependent; this makes unit testing a sequential process.

Two steps with freq.nlevels and scl functions

Overview

This section contains these topics.

- fetch n-obs, n-vars
 - fetch n-levels
 - describe output data structure
 - calculate cardinality ratios and mean
 - assign cr-type
-

fetch n-obs, n-vars

```
%let _dsid  = %sysfunc(open (&libname..&memname,i));
%let _n_obs = %sysfunc(attrn(&_dsid,nobs));
%let _n_vars = %sysfunc(attrn(&_dsid,nvar));
%let _rc    = %sysfunc(close(&_dsid));
```

fetch n-levels

```
ODS select none; * noprint;
PROC freq data   = &libname..&memname nlevels;
      ods output
            nlevels= list_names
                  (keep = tablevar      nlevels
                    rename=(tablevar= name nlevels= n_levels));
run;
ODS select all;
```

describe data structure

```
DATA &syslast;
  attrib memname    length = $32
         varnum     length = 8
         cr_type    length = $ %length(n-levels=1)
         card_ratio length = 8    %*range=(0:1];
                           format = bestd7.5
         n_levels   length = 8
         name       length = $32
         type       length = $ 1
         length     length = 8
         format     length = $49
         label      length = $256;
```

calculate cardinality ratios and mean

```
array _cr(&n_vars);

** loop: for each row, calculate cardinality ratio;
do _i = 1 to dim(_cr);
  set &syslast (keep = n_levels) point = _i;
  _cr(_i) = n_levels / &n_obs;          %*global mvar;
end;

***** calculate mean for select card_ratio to cr_type;
_mean_cr = mean(of _cr(*));
```

fetch variable information

```
_dsid = open("&libname..&memname");
do _i = 1 to dim(_cr);
  set &syslast point = _i;
  *...;
  card_ratio = _cr(_i);
```

Notes: The assumption is data and freq.n-levels are in varnum order.

assign cr-type

```
select;
  when(n_levels eq 1) cr_type = 'n-levels=1';
  when(card_ratio eq 1) cr_type = '.unique';
  when(card_ratio gt _mean_cr) cr_type = 'many';
  otherwise                cr_type = 'few';
end;
output;
```

citations: There are a number of functions for arrays, this paper [38] shows call sortn.

Previous papers on cardinality ratio include [32] [33], [35]

make-list-cr-types-2-steps

```
*name: make-list-cr-types-2-steps.sas;
*called by: make-list-cr-types-2-steps-memname.sas
ODS select none; * noprint;
PROC freq data = &libname..&memname nlevels;
ods output
  nlevels= list_names
           (keep = tablevar          nlevels
            rename=(tablevar= name nlevels= n_levels));
run;
ODS select all;
%let _dsid = %sysfunc(open (&libname..&memname,i));
%let _n_obs = %sysfunc(attrn(&_dsid,nobs));
%let _n_vars = %sysfunc(attrn(&_dsid,nvar));
%let _rc = %sysfunc(close(&_dsid));
DATA cr_types(label = "&memname nobs=&_n_obs nvars=&_n_vars");
  attrib memname      length = $32
         varnum       length = 8
         cr_type      length = $ %length(n-levels=1)
         card_ratio   length = 8    %*range=(0:1);
                           format = bestd7.5
         n_levels     length = 8
         name         length = $32
         type         length = $ 1
         length       length = 8
         format       length = $49
         label        length = $256;
  array _cr(&_n_vars);
  drop _:; * _temp vars;
  retain memname "&memname";

  ** loop: for each row, calculate cardinality ratio;
  do _i = 1 to dim(_cr);
    set &syslast (keep = n_levels) point = _i;
    _cr(_i) = n_levels / &_n_obs;    *global macro variable;
  end;

  ***** calculate mean for select card_ratio to cr_type;
  _mean_cr = mean(of _cr(*));

  _dsid = open("&libname..&memname");
  do _i = 1 to dim(_cr);
    set &syslast point = _i;
    varnum = varnum (_dsid,name );
    type = lowercase(vartype (_dsid,varnum));
    length = varlength(_dsid,varnum);
    format = varfmt (_dsid,varnum);
    label = varlabel (_dsid,varnum);
    card_ratio = _cr(_i);
    select;
      when(n_levels eq 1 ) cr_type = 'n-levels=1';
      when(card_ratio eq 1 ) cr_type = '.unique';
      when(card_ratio gt _mean_cr) cr_type = 'many';
      otherwise cr_type = 'few';
    end;
    output;
  end;
  _rc = close(_dsid);
stop;
run;
```

Notes: The freq.nlevels data set is output in varnum order so _i is varnum.

One step with scl functions and dosubl

Overview

This is the list of topics in this section.

- make list cr types for memname
- proc freq dosub
- programming issues
- make list cr types 1 step
- calculate card-ratios and mean
- assign cr-type

make list cardinality types for memname

```
*name:      make-list-cr-types-1-step-memname.sas;
* see also make-list-cr-types-1-step-libname.sas;
%let libname = sashelp;
%let memname = class;
%let memnum = 0;
%let out_lib = library;
%let out_lib = work;

%include 'make-list-cr-types-1-step.sas';

PROC print data = &syslast noobs;
    title3 "Cardinality Ratios, Types of &libname..&memname";
run;
```

proc freq dosub

```
%macro proc_freq_dosub
    (data      = &libname..&memname /*global*/
     ,name      =
     ,type      =
     ,varnum     =
     ,memnum     = 0
     ,out_lib    = work );
%put echo &sysmacroname &=name &=type &=varnum &=memnum;
PROC freq data = &data;
    tables    &name / noprint
              out = &out_lib..freq_&memnum._&varnum
                  (label = "&name"
                   rename = (&name = valu_&type));
run;
%put echo &sysmacroname &=sysnobs;
%global _n_levels;
*robust: avoid divide by zero in calling program;
*** sysnobs returns -1 for data-set w/nobs=0;
%if &sysnobs gt 0 %then
    %let _n_levels = &sysnobs;
%else %let _n_levels = 1;
%mend proc_freq_dosub;
```

programming issues

In the first edition of this macro I assumed that the global system macro variable `sysnobs` would be updated. As you can see in the log the statement `%put echo &sysmacroname &=sysnobs;` prints the number of observations of the data set that the procedure produced. However, in program `make-list-cr-types-1-step` the fetch of `sysnobs` fails.

`_nlvl(varnum)=input(symget('sysnobs'),32.);`

Thus the solution shown here: allocate a global macro variable `n-levels` and conditionally set its value to `sysnobs`.

citations: Function `dosubl` is described in Langston [47].

make list cr types 1 step

```

* name      : make-list-cr-types-1-step.sas;
* called by: make-list-cr-types-1-step-memname.sas;
* called by: make-list-cr-types-1-step-libname.sas;
%put echo &libname..&memname &=memnum &=out_lib;
%let _dsid  = %sysfunc(open (&libname..&memname));
%let _n_obs = %sysfunc(attrn(&_dsid,nobs));
%let _n_vars = %sysfunc(attrn(&_dsid,nvar));
%let _rc    = %sysfunc(close(&_dsid));
DATA &out_lib..list_cr_types_&memnum
  (label = "memname=&memname,obs=&_n_obs,vars=&_n_vars");
  attrib memname      length = $32
         memnum       length = 8 label = 'mem num'
         varnum       length = 8 label = 'var num'
         cr_type      length = $ %length(n-levels=1)
                     label = 'card. ratio type'
         card_ratio   length = 8 label = 'card. ratio'
                     format = bestd7.5
         n_levels     length = 8 label = 'n-levels'
         name         length = $32
         type         length = $ 1
         length       length = 8
         format       length = $49
         label        length = $256;
  array _cr (&_n_vars) ;
  array _name (&_n_vars) $32;
  array _nlvl (&_n_vars) ;
  array _type (&_n_vars) $ 1;
  drop _.;
  retain memname "&memname" memnum &memnum;
  _dsid = open("&libname..&memname");
  do varnum = 1 to dim(_name);
    _name(varnum)= varname(_dsid,varnum) ;
    _type(varnum)= lowcase(vartype(_dsid,varnum));
    _rc = dosubl(catt('%proc_freq_dosub(name=',_name(varnum)
                      ',type=',_type(varnum)
                      ',varnum=', varnum
                      ',memnum=&memnum,out_lib=&out_lib)'));
    _nlvl(varnum) = input(symget('n_levels'),32.);
    if _nlvl(varnum) and &_n_obs then
      _cr(varnum) = _nlvl(varnum)/&_n_obs;
    else _cr(varnum) = 1/&sysmaxlong;*small and non-zero;
  end;
  ***** calculate mean for select card_ratio to cr_type;
  _mean_cr = mean(of _cr(*));
  do varnum = 1 to dim(_name);
    name      = _name      ( varnum);
    card_ratio = _cr        ( varnum);
    n_levels  = _nlvl      ( varnum);
    type      = _type      ( varnum);
    length    = varlength(_dsid,varnum);
    format    = varfmt     (_dsid,varnum);
    label     = varlabel   (_dsid,varnum);
  select;
    when(n_levels eq 1) cr_type = 'n-levels=1';
    when(card_ratio eq 1) cr_type = '.unique';
    when(card_ratio gt _mean_cr) cr_type = 'many';
    otherwise cr_type = 'few';
  end;
  output;
end;
_rc = close(_dsid);
run;

```

Cardinality types of all memnames in a libref

Overview

Notice that program `make-list-cr-types-1-step-memname` on page 23 has a parameter `memnum`.

Routine `make-list-cr-types-1-step` on page 24 creates a report with a unique name `make-list-cr-types-*` and passes this on to macro `proc-freq-dosub` on page 23 which creates a unique name for each output data set `freq-(memnum)-(varnum)` from the frequency procedure.

This section has these topics.

- `make list cr types for libname`
- `list of data sets`

`make list cr types for libname`

```
* name:      make-list-cr-types-1-step-libname.sas;
* see also  make-list-cr-types-1-step-memname.sas;
%let libname = sashelp; *17K files!  time= 5+ hours!;
%let libname = library;
*let libname = my_lib;
%let out_lib = library;
%let out_lib = work;

%include 'make-list-memnames-contents.sas';
%let cx_data    = &syslast(keep = nobis memname memnum
                        where =(nobis));
%let cx_include = 'make-list-cr-types-1-step.sas'/source2;
%include 'cx-include.sas'/nosource2;

PROC sql; select memname, nobis, nvar, memlabel length=32
               from dictionary.tables
               where libname eq "%upcase(&out_lib)"
                  and memtype eq 'DATA';
               quit;
run;
```

`list of data sets`

Member Name	Number of Physical Obs	Number of Vars	Data Set Label
-----	-----	-----	-----
FREQ_1_1	9	3	locale
FREQ_1_2	184	3	key
FREQ_1_3	1	3	lineno
FREQ_1_4	1345	3	text
FREQ_2_1	1	3	locale
...			
LIST_CR_TYPES_1	4	11	memname=AACOMP,obs=1544,vars=4
LIST_CR_TYPES_2	4	11	memname=AARFM,obs=61,vars=4
...			

citations: Programs in this paper are available on [10].

Summary

Conclusion

A SAS data set is an example of a list, the attribute items of which — *n-obs* and *n-vars* — contain information that can be used to process each variable and provide additional attributes of that variable — *n-levels* — that can be compared not only to the other variables but also to the data set set attribute, *n-obs*. The new attributes of *cardinality ratio* and *cardinality type* explain the three major categories of variables: *row-identifier: unique*, *classification: few*, and *analysis: many*.

This information can then be used to coordinate other programming tasks.

future work

This paper describes a set of tools which can be used to analyze a data set or all data sets in a libref. Here are my plans for future work.

- make list of user-defined formats and convert data set of frequency of variable to frequency of variable with format, [15]
- for card-type equal many and type equal numeric,
 - find macro `freqall` in [20] and modify it into macro `freq_hilo`; produce a report of all the high and low values of each numeric
 - write a macro `smry_from_freq` where the summary procedure reads the frequency output data set with `weight count`;
- for few: calculate `max-length-char` and write a report to stack (set) or append into one data set all the categorical variables
- *all(few) x each(many)*: make a list of the cr-type eq 'few' variables and create a program to do a cross-tabulation and summary of each cr-type eq 'many' variables, where type eq numeric per algorithm of Raithel [48]
- exclude cr-type of `UNIQUE` and `N-LEVELS=1` from calculation of `mean(cr)`

program downloads

The SAS community wikipedia has pages for the programs shown here. See also the category macros by Ron Fehd.

- macro arrays [11] 1998, v1, call `symputx` [17] 2004, v2, sql `select into` [1] 2007, v3, a function and procedure
- making lists contains `make-list-memnames`, `make-list-names` and `ml-name-x`:
`make-list-names-cardinality-ratio-2-steps`, page 22
- loops `cx-include`, formerly `callxinc` 2008 and `cxinclude` 2012
`call-macro` `call-text`
- cardinality ratio `database vocabulary` `summarize each var`
- program design [19], [22], [39] and [30]

citations:

Acknowledgements

Quentin McMullen and Rick Langston reviewed the next-to-last draft and provided helpful commentary. Art Carpenter shared the idea of a list as a control data set.

Author Information

Ronald J. Fehd

`Ron.Fehd.macro.maven@gmail.com`

About the author:

sco.wiki

http://www.sascommunity.org/wiki/Ronald_J._Fehd

LinkedIn

www.linkedin.com/Ronald.Fehd

affiliation

Stakana Analytics, Senior Maverick

also known as

macro maven on SAS-L, Theoretical Programmer

References

- [1] Editor R.J. Fehd. Macro array package. In *sasCommunity.org*, 2007. URL http://www.sascommunity.org/wiki/Macro_Array_Package. macro array is both function to return dimension and procedure to create array of macro variables; solves problem of creating local macro variables inside calling macro.
- [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. ListMcat: list macro catalog. In *sasCommunity.org*, 2009. URL http://www.sascommunity.org/wiki/ListMcat_List_Macro_Catalog. program to list macro definitions in work and sasmstored catalogs.
- [4] Editor R.J. Fehd. How to use sql select into for list processing. In *sasCommunity.org*, 2010. URL http://www.sascommunity.org/wiki/How_to_Use_Proc_SQL_select_into_for_List_Processing. review of sql dictionaries and use of select text and values into :mvar.
- [5] Editor R.J. Fehd. Macro Call-Macro. In *sasCommunity.org*, 2012. URL http://www.sascommunity.org/wiki/Macro_CallMacro. using SCL functions to read a data set and call macros.
- [6] 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.
- [7] Editor R.J. Fehd. Macro do-loop. In *sasCommunity.org*, 2012. URL http://www.sascommunity.org/wiki/Macro_Do-Loop. macro function to return tokens inside a statement.
- [8] Editor R.J. Fehd. Macro loops with dates. In *sasCommunity.org*, 2013. URL http://www.sascommunity.org/wiki/Macro_Loops_with_Dates. example macros, programs and updates.
- [9] Editor R.J. Fehd. Macro Exist. In *sasCommunity.org*, 2014. URL http://www.sascommunity.org/wiki/Macro_Exist. code to check existence of catalogs, datas, filenames, filerefs, librefs.
- [10] Editor R.J. Fehd. Cardinality ratios and types. In *sasCommunity.org*, 2017. URL http://www.sascommunity.org/wiki/Cardinality_Ratios_and_Types.
- [11] Ronald Fehd. %Array: construction and usage of arrays of macro variables. In *SouthEast SAS Users Group Conference Proceedings*, 1996. URL <http://www.lexjansen.com/sesug/1996/SESUG96027.pdf>. 5 pp.; uses call symput.
- [12] Ronald Fehd. %Checkall: A macro to produce a frequency of response data set from multiple-response data. In *SouthEast SAS Users Group Conference Proceedings*, 1996. URL <http://www.lexjansen.com/sesug/1996/SESUG96080.pdf>. 6 pp.;
- [13] Ronald Fehd. %Showcomb; a macro to produce a data set with frequency of combinations of responses from multiple-response data. In *SouthEast SAS Users Group Conference Proceedings*, 1996. URL <http://www.lexjansen.com/sesug/1996/SESUG96081.pdf>. 4 pp.;
- [14] Ronald Fehd. %Array: construction and usage of arrays of macro variables. In *SAS Users Group International Annual Conference Proceedings*, 1997. URL <http://www2.sas.com/proceedings/sugi22/CODERS/PAPER80.PDF>. 4 pp.; uses call symput.
- [15] Ronald Fehd. %Freq1Var: Frequency of one variable with format, a macro to standardize proc freq output data sets. In *SAS Users Group International Annual Conference Proceedings*, 1999. URL <http://www2.sas.com/proceedings/sugi24/Posters/p234-24.pdf>. Posters, 3 pp.
- [16] Ronald Fehd. Array: Construction and usage of arrays of macro variables. In *NorthEast SAS Users Group Conference Proceedings*, 2003. URL www.lexjansen.com/nesug/nesug03/cc/cc015.pdf. Coders Corner, 6 pp.; sql select into.
- [17] Ronald Fehd. Array: construction and usage of arrays of macro variables. In *SouthEast SAS Users Group Conference Proceedings*, 2004. URL <http://analytics.ncsu.edu/sesug/2004/SY03-Fehd.pdf>. 4 pp.; uses sql select into.
- [18] Ronald Fehd. A SASautos Companion: Reusing macros. In *SAS Users Group International Annual Conference Proceedings*, 2005. URL <http://www2.sas.com/proceedings/sugi30/267-30.pdf>. Tutorials, 12 pp.; topics: autocall, autoexec, configuration file (sasv9.cfg), compiled and stored macros, masking, options (mautosource, mstored, sasmstore), program reuse, sasautos (environment variable, filename, option); info: autoexec examples, utility program ListMcat: show same-named macros in different catalogs.
- [19] Ronald J. Fehd. Building reusable programs as includes or macros. In *SouthEast SAS Users Group Conference Proceedings*, 2005. URL http://analytics.ncsu.edu/sesug/2005/CC06_05.PDF. Applications, 1 pp.
- [20] Ronald J. Fehd. Journeymen's tools: Data review macro FreqAll: Using proc sql list processing with dictionary.columns to eliminate macro do loops. In *NorthEast SAS Users Group Conference Proceedings*, 2006. URL <http://www.nesug.org/proceedings/nesug06/cc/cc14.pdf>. Coders' Corner, 9 pp.; arrays of macro variables, data structure.

- [21] Ronald J. Fehd. FreqLibname: a data review routine for all memnames in a libname. In *NorthEast SAS Users Group Conference Proceedings*, 2007. URL <http://www.nesug.org/proceedings/nesug07/cc/cc11.pdf>. Coders' Corner, 22 pp.; topics: comments, documentation, quality, theory, replacing macros with call execute of parameterized include files, saving procs freq and summary output data set; info: complete test suite of modules, routines, and subroutines, getting mode from proc freq.
- [22] Ronald J. Fehd. Writing testing-aware programs that self-report when testing options are true. In *NorthEast SAS Users Group 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.
- [23] Ronald J. Fehd. Do which? loop, until or while? a review of data step and macro algorithms. In *SAS Global Forum Annual Conference Proceedings*, 2007. URL <http://www2.sas.com/proceedings/forum2007/067-2007.pdf>. Coders Corner, 9 pp.; info: comparison of boolean logic used in until and while, do until(last.var); if skip-condition then continue, if exit-condition then leave, loop-repeat with go-to; bibliography.
- [24] 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 paper in ApDev; code available at sas.community.org, SmryEachVar_A.Data.Review.Suite.
- [25] Ronald J. Fehd. 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*, 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.
- [26] Ronald J. Fehd. Call execute a parameterized include. In *SAS Community Wiki*, 2008. URL http://www.sascommunity.org/wiki/Call_Execute_Parameterized_Include. url for this article.
- [27] 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 SAS Users Group 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.
- [28] Ronald J. Fehd. Using functions Sysfunc and Ifc to conditionally execute statements in open code. In *Western Users of SAS Software Annual Conference Proceedings*, 2009. URL www.lexjansen.com/wuss/2009/cod/C0D-Fehd.pdf. Coders Corner, 11 pp.
- [29] Ronald J. Fehd. How to use proc SQL select into for list processing. In *SouthEast SAS Users Group Conference Proceedings*, 2010. URL <http://analytics.ncsu.edu/sesug/2010/H0W06.Fehd.pdf>. Hands On Workshop, 40 pp.; topics: writing constant text, and macro calls, using macro %do loops; references.
- [30] Ronald J. Fehd. Macro design ideas: Theory, template, practice. In *MidWest SAS Users Group Annual Conference Proceedings*, 2013. URL <http://www.mwsug.org/proceedings/2013/00/MWSUG-0002.pdf>. 21 pp.; topics: logic, quality assurance, testing, style guide, documentation, bibliography.
- [31] Ronald J. Fehd. Writing macro do loops with dates from then to when. In *MidWest SAS Users Group Annual Conference Proceedings*, 2013. URL <http://analytics.ncsu.edu/sesug/2013/CC-03.pdf>. 20 pp.; topics: dates are integers, formats and functions to convert date references to integers, calculations, bibliography.
- [32] Ronald J. Fehd. Database vocabulary: Is your data set a dimension (lookup) table, a fact table or a report? In *SouthEast SAS Users Group Conference Proceedings*, 2013. URL <http://www.analytics.ncsu.edu/sesug/2013/CC-04.pdf>. 9 pp.; cardinality ratio, composite key, database design, foreign key, grain, nlevels, normal forms, primary key, relational database, snapshots: accumulating or periodic.
- [33] Ronald J. Fehd. Data review information: N-levels or cardinality ratio. In *SAS Global Forum Annual Conference Proceedings*, 2013. URL <http://support.sas.com/resources/papers/proceedings13/299-2013.pdf>. Statistics and Data Analysis, 6 pp.; using proc freq nlevels and nobs to calculate cardinality ratio — range in (0:1) — of a variable to determine its type in (continuous, discrete, unique, worthless).
- [34] Ronald J. Fehd. List processing macro call-macro. In *MidWest SAS Users Group Annual Conference Proceedings*, 2014. URL www.mwsug.org/proceedings/2014/BB/MWSUG-2014-BB04.pdf. 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.
- [35] Ronald J. Fehd. Using cardinality ratio for fast data review. In *MidWest SAS Users Group Annual Conference Proceedings*, 2014. URL <http://www.lexjansen.com/mwsug/2014/RF/MWSUG-2014-RF04.pdf>. Rapid Fire, 16 pp.; continuous, database, dimensionless, discrete, fact, frequency, keys: foreign or primary, nlevels, number of observations (nobs), unique.
- [36] Ronald J. Fehd. Macro design ideas, theory, template, practice. In *Western Users of SAS Software Annual Conference Proceedings*, 2014. URL http://www.lexjansen.com/wuss/2014/93_Final_Paper_PDF.pdf. Coders Corner, 21 pp.

- [37] Ronald J. Fehd. An autoexec companion, allocating location names during startup. In *MidWest SAS Users Group Annual Conference Proceedings*, 2015. URL <http://www.lexjansen.com/mwsug/2015/BB/MWSUG-2015-BB-10.pdf>. Beyond Basics, 15 pp.; autocall macros, global symbol table, filerefs, librefs, cexist catalogs, exist data set, sasautos.
 - [38] Ronald J. Fehd. Reading a column into a row to count n-levels, calculate cardinality ratio and create frequency and summary output in one step. In *MidWest SAS Users Group Annual Conference Proceedings*, 2015. URL <http://www.lexjansen.com/mwsug/2015/BB/MWSUG-2015-RF-04.pdf>. Rapid Fire, 10 pp.
 - [39] Ronald J. Fehd. Applications design and development, a case study of the EDA summarize-each-variable suite. In *SouthEast SAS Users Group Conference Proceedings*, 2015. URL http://www.lexjansen.com/sesug/2015/96_Final_PDF.pdf. Hands On Workshops, 22 pp.
 - [40] Ronald J. Fehd. A sysparm companion, passing values to a program from the command line. In *MidWest SAS Users Group Annual Conference Proceedings*, 2016. URL <http://www.mwsug.org/proceedings/2016/TT/MWSUG-2016-TT04.pdf>. Tools of Trade, 8 pp.; shows use of sysparm as macro variable and option which can be assigned value on command line in batch programs; program parse-sysparm parses a list of comma-separated values (csv) of form var1=value1,var2=value2,...,varN=valueN into macro variables.
 - [41] Ronald J. Fehd. List processing macro call-text. In *MidWest SAS Users Group Annual Conference Proceedings*, 2016. 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.
 - [42] Ronald J. Fehd. True is not false: Evaluating logical expressions. In *SouthEast SAS Users Group Conference Proceedings*, 2016. URL <http://www.lexjansen.com/sesug/2016/BB-186.pdf>. Beyond the Basics, 9 pp.; Boolean algebra, Boolean logic, De Morgan's law, evaluation, logical operators, sql joins.
 - [43] Ronald J. Fehd. Macro code to test existence of various objects. In *SouthEast SAS Users Group Conference Proceedings*, 2016. URL <http://www.lexjansen.com/sesug/2016/CC-187.pdf>. 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.
 - [44] 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.
 - [45] Christian Heilmann and Staff W3C. Programming — the real basics. In *W3C*, 2011. URL https://www.w3.org/wiki/Programming_-_the_real_basics. variables, types of variables, conditions, loops, summary.
 - [46] Ralph Kimball and Margy Ross. *The Data Warehouse Toolkit, The Complete Guide to Dimensional Modeling, Second Edition*. John Wiley & Sons, Inc., New York, 2002. URL <http://www.kimballgroup.com/html/booksDWT2.html>. subtitle: Complete Guide to Dimensional Modeling; 17 chap., 387 pp., glossary: 29 pp., index: 18 pp.
 - [47] Rick Langston. Submitting SAS(R) code on the side. In *SAS Global Forum Annual Conference Proceedings*, 2013. URL <http://www2.sas.com/proceedings/sugi25/25/sy/25p290.pdf>.
 - [48] Michael A. Raithel. Summarizing impossibly large sas data sets for the data warehouse server using horizontal summarization. In *SAS Users Group International Annual Conference Proceedings*, 2000. URL <http://www2.sas.com/proceedings/sugi25/25/sy/25p290.pdf>.
 - [49] Pamela Statz. Get started with programming. In *Wired*, Feb 2010. URL https://www.wired.com/2010/02/get_started_with_programming/. variables, scope of variables, conditionals, loops, functions, tips.
-