PharmaSUG 2023 — Paper AP-015

Using SQL Dictionaries to Research the Global Symbol Table

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

Abstract

description The sql procedure in SAS® software provides a number of dictionaries that can be used to research entries in the global symbol table. These dictionaries include lists of dataset and variable names, option values, and catalog entries for format values and macro definitions.

purpose This paper provides example programs to research values in the global symbol table assigned by the global statement options, procedure output from the format procedure, and macro definitions.

The sql procedure can also be used to create lists of objects for list processing: list of variable names, or dataset names.

audience all levels

sql: dictionary.dictionaries, dictionary.catalogs, dictionary.columns,
dictionary.tables, dictionary.options;
procedures: catalog, format, contents, print, sql;
options: format: fmtsearch,
macro definitions: autocall: mautosource + sasautos,
compiled+stored: mstored + sasmstore

In this paper:

List of programs:

Introduction

overview

Each sql dictionary is an associative array: each row has a key — a row-identifier, which may be one or more columns (composite key) — and one or more labels containing information about the key. Our task throughout is to first examine the data structure, identify the column or columns that are the key, and then discover how to use the table.

This is the list of topics:

- describe libref.memname
- syntax of sql statements
- syntax of select
- index of usage of sql clauses

describe libref.memname

Program 1 is thesyntax of the describe statement; this program is used throughout this paper.

Program 1 sql-describe-libref-memname.sas

```
proc sql; describe table libref.memname; quit;
```

syntax of sql statements

```
proc sql;
proc sql noprint;
         create    table table-name as
                   query-expression
                    <order by order-by-item
                     <, ...    order-by-item>>;
         describe  table table-name <, ... table-name>;
         drop      table libref.memname;
         select    <distinct> object-item
                   <function>(object-item) <as new-name>
                   <, ...      object-item>
           into   :macro-variable <separated by ' '>
                  :macro-variable-A, :macro-variable-B
                  :macro-variable1 - :macro-variable&sysmaxlong
                  :macro-variable1 - %*short form: only hyphen;
           from    libref.memname
          where    column-char  eq 'value'
            and    column-char2 eq "%upcase(&mvar)"
            or     column-num   eq <num-value>
          group by group-by-item
                   <, ... group-by-item>>
          having   sql-expression
          order by order-by-item
                   <, ... order-by-item>>;
         quit;
```

syntax of select

The keyword `select` has one required clause, `from`, and five optional clauses: `into`, `where`, `group by`, `having`, and `order by`, which may be viewed conceptually in this hierarchy:

```
select                   col_1 <,col_2, ..., col_N>
       into             :mvar    <separated by '<dlm>'>
       from
                where    col_1 eq 'value'
       group by          col_1 <,col_2, ..., col_N>
                having   col_1 eq 'value'
       order by          col_1 <,col_2, ..., col_N>
```

index of usage of sql clauses

Note: in this table `name` is a column (sql) or variable (dataset) name.

object	clause	pg.
name	as new_name label = 'var label'	4
	count(name) as new_name	13
	distinct name	4
	group by name	13
name	into :mvar separated by ';'....	5
name	like 'prefix%'..................	18
name eq	"%upcase(&mvar)"	7
	order by name	18

How many sql dictionaries?

The composite key of `dictionary.dictionaries` is columns `memname + name + (name or varnum)`.
Refer to the log saved from program 3, shown on pg. 5,
or use this program to write notes to the log.

```
proc sql; describe table dictionary.dictionaries; quit;
```

log

```
NOTE: SQL table DICTIONARY.DICTIONARIES was created like:
   create table DICTIONARY.DICTIONARIES
  (memname  char(32)  label='Member Name',
   memlabel char(256) label='Data Set Label',
   name     char(32)  label='Column Name',
   type     char(4)   label='Column Type',
   length   num       label='Column Length',
   npos     num       label='Column Position',
   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'
```

Program 2 provides a list of the `SQLOBS=32` dictionaries. Note that values of datasets (memname) and variables (name) within all dictionaries are in upper case; they are `lowcase` here for readability.

Program 2 sql-d-dictionaries-memnames-list.sas

```
proc sql; select distinct lowcase(memname)
                 as memname label = 'Member Name'
          from   dictionary.dictionaries;
          quit;
          %put &=sqlobs;
```

lst

```
Member Name
----------------------
catalogs               <---<<<
check_constraints
columns                <---<<<
constraint_column_usage
constraint_table_usage
dataitems
destinations
dictionaries           <---<<<
engines
extfiles
filters
formats
functions
goptions
indexes
infomaps
libnames
locales
macros
members
```

```
options                 <---<<<
prompts
promptsxml
referential_constraints
remember
styles
tables                  <---<<<
table_constraints
titles
views
view_sources
xattrs
```

notes: The dictionaries reviewed in this paper are:
catalogs, columns, dictionaries, options, and tables.

describe each dictionary

Programs 3 and 4 assemble the statements:

```
  describe table dictionary.<memname>;
```

for each of the sql dictionaries.

Program 3 sql-d-dictionaries-memname-describe-each-1.sas, select into :list

```
proc sql; select  distinct catt(
                 'describe table dictionary.',memname,';')
            into :list       separated by ' '
            from  dictionary.dictionaries;
         &list
          quit;
%symdel   list;
```

notes: **&list** submits the statements in the macro variable `list`
%symdel erases the macro variable `list` from the global symbol table
Program 3 shows the essence of list processing: using value of the key — `memname` — in a loop and enclosing that value in constant text:
prefix = `describe table dictionary.` and suffix = `semicolon;`.
Program 3 is **fragile** for two reasons:

default length : of the temporary variable varies by operating system;
this is a soft limit: it can be fixed;
while testing you will see that the text is truncated;
solution: `select ’...’  length = ???  into :list`

maximum length : of the macro variable `list` is limited to
`(2**16)-2 = 65,534;`
minus two refers to the delta-quotes surrounding the value;
this is a hard limit: it cannot be fixed;
see program 4 for the array of macro variables solution

Program 4 is the workaround for either of the issues of program 3:
length of text, or too much text in the macro variable.
It creates a sequentially-numbered set of macro variables — an array of macro variable values: `item1 ... itemN`, which are then referenced in a loop. The quirky reference `&&item&i` resolves to one of the `item?` series of macro variables.

Program 4 sql-d-dictionaries-memname-describe-each-2.sas, select into :item1 -

```
%macro describe_each();     *array of macro variables;
proc sql; select  distinct lowcase(memname)
            into :item1 -           %*item1 ... itemN;
            from  dictionary.dictionaries;
          %put echo n(items): &=sqlobs   ;
          %do i = 1 %to &sqlobs;
              %let value = &&item&i;
              %put echo &=value;
              describe table dictionary.&value;
              %*---- constant text ----*______;
              %end;
          quit;
%mend describe_each;
     %describe_each()
```

notes:
lst

Either of programs 3 or 4, produces this output:

```
describe table dictionary.CATALOGS;
describe table dictionary.CHECK_CONSTRAINTS;
describe table dictionary.COLUMNS;
...
describe table dictionary.VIEWS;
describe table dictionary.VIEW_SOURCES;
describe table dictionary.XATTRS;
```

! →
log

Save this log! and review to find the description of each dictionary.

```
NOTE: SQL table DICTIONARY.CATALOGS was created like:
NOTE: SQL table DICTIONARY.CHECK_CONSTRAINTS was created like:
NOTE: SQL table DICTIONARY.COLUMNS was created like:
...
NOTE: SQL table DICTIONARY.VIEWS was created like:
NOTE: SQL table DICTIONARY.VIEW_SOURCES was created like:
NOTE: SQL table DICTIONARY.XATTRS was created like:
```

Dataset, variables: columns in tables

overview

This is the list of topics in this section:

- columns: variable names
  - making list of (variable) names, sql
- tables: dataset names
  - making list of (dataset) memnames, sql
- compare to proc contents
  - make list names, contents
  - make list memnames, contents

columns: variable names

Refer to the log saved from program 3, shown on pg. 5,
or use this program to write notes to the log.

```
proc sql; describe table dictionary.columns; quit;
```

log

```
NOTE: SQL table DICTIONARY.COLUMNS was created like:
   create table DICTIONARY.COLUMNS
  (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',
   npos       num       label='Column Position',
   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',
```

notes: Table `dictionary.columns` has a composite key:
columns `libname + memname + (name or varnum)`.

making list of (variable) names

Program 5 creates a dataset of variable names from sql dictionary.columns.

Program 5 make-list-names-sql.sas

```
%let libname = sashelp;
%let memname = class;
proc sql; create   table list_names as
          select   libname, memname, varnum, name, type
            from   dictionary.columns
            where  libname eq "%upcase(&libname)"
              and  memname eq "%upcase(&memname)"
              and  memtype eq 'DATA';
          describe table  &syslast;
          select   * from &syslast;
          quit;
```

log

```
NOTE: SQL table WORK.LIST_NAMES was created like:
   create table WORK.LIST_NAMES
  (libname char(8)  label='Library Name',
   memname char(32) label='Member Name',
   varnum  num      label='Column Number in Table',
   name    char(32) label='Column Name',
   type    char(4)  label='Column Type'
```

lst

```
Library    Member   Column   Column   Column
Name       Name     Number   Name     Type
-------   -------   ------   ------   ------
SASHELP   CLASS          1   Name     char
SASHELP   CLASS          2   Sex      char
SASHELP   CLASS          3   Age      num
SASHELP   CLASS          4   Height   num
SASHELP   CLASS          5   Weight   num
```

notes: Column `varnum` is a *natural key*; its values are integers and are in (1–

n(columns)). Column type values are in ('char','num'); compare to the contents procedure output, which are in (1:char,2:num), and shown in program make-list-names-contents.sas, on pg. 10.

cardinality ratio and type

This list of variable names can be expanded by calculating cardinality ratios and types; the program is shown in
Fehd, "Calculating Cardinality Ratio in Two Steps".

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

tables: dataset names

Refer to the log saved from program 3, shown on pg. 5,
or use this program to write notes to the log.

```
proc sql; describe table dictionary.tables; quit;
```

log

```
NOTE: SQL table DICTIONARY.TABLES was created like:
   create table DICTIONARY.TABLES
  (libname       char(8)   label='Library Name',
   memname       char(32)  label='Member Name',
   memtype       char(8)   label='Member Type',
   dbms_memtype  char(32)  label='DBMS Member Type',
   memlabel      char(256) label='Data Set Label',
   typemem       char(8)   label='Data Set Type',
   crdate        num       label='Date Created',
   modate        num       label='Date Modified',
   nobs          num       label='Number of Physical Observations',
   obslen        num       label='Observation Length',
   nvar          num       label='Number of Variables',
```

notes: Notice that column nvar is singular.
Table dictionary.columns has a composite key:
columns libname + memname.

making list (dataset) memnames, sql

Program 6 creates a dataset from sql dictionary.tables.

Program 6 make-list-memnames-sql.sas

```
%let libname = sashelp;
proc sql; create   table list_memnames as
          select   libname, memname, nobs, nvar
            from   dictionary.tables
            where  libname eq "%upcase(&libname)"
              and  memtype eq 'DATA';
          describe table  &syslast;
          select   * from &syslast;
          quit;
```

log

```
NOTE: SQL table WORK.LIST_MEMNAMES was created like:
   create table WORK.LIST_MEMNAMES
  (libname char(8)  label='Library Name',
   memname char(32) label='Member Name',
   nobs    num      label='Number of Physical Observations',
   nvar    num      label='Number of Variables'
```

lst

```
                   Number of
Library   Member   Physical      Number of
Name      Name     Observations  Variables
SASHELP   AACOMP           2020          4
SASHELP   AARFM             130          4
SASHELP   ADSMSG            426          6
...
SASHELP  ZIPCODE          41140         21
SASHELP  ZIPMIL             560         21
SASHELP  ZTC              18161          6
SASHELP  _CMPIDX_            44         13
```

compare to proc contents

Program 7 creates a table of (variable) names using the contents procedure.

Program 7 make-list-names-contents.sas

```
%let data = sashelp.class;
PROC contents data = &data         noprint
              out  = list_names
                    (keep = libname memname varnum name type
                             memtype
                     where= (memtype eq 'DATA'));
run;
proc sql;  describe table &syslast; quit;
proc print data = &syslast(drop = memtype);
run;
```

log

```
NOTE: SQL table WORK.LIST_NAMES was created like:
   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',
   VARNUM  num      label='Variable Number',
   MEMTYPE char(8)  label='Library Member Type'
```

lst

```
Obs   LIBNAME   MEMNAME   NAME     TYPE   VARNUM
---   -------   -------   ------   ----   ------
  1   SASHELP   CLASS     Age        1       3
  2   SASHELP   CLASS     Height     1       4
  3   SASHELP   CLASS     Name       2       1
  4   SASHELP   CLASS     Sex        2       2
  5   SASHELP   CLASS     Weight     1       5
```

notes: The contents procedure output table is sorted alphabetically by column: `name`; note that the column `varnum` is not in ascending order.
Column `type` is in (1:num,2:char); compare to dictionary.columns.type in ('char','num') shown above in program make-list-names-sql.sas, pg. 7.

make list memnames, contents

Program 8 creates a table of memnames and attributes using the contents procedure.

Program 8   make-list-memnames-contents.sas

```
%let libname = sashelp;
PROC contents data = &libname.._all_        noprint
              out  = list_memnames_contents
                    (keep = libname memname nobs memlabel
                             memtype varnum
                     where= (memtype eq 'DATA'));
run;
proc sql; create table list_memnames_&libname as
          select     distinct memname, nobs,
                 max(varnum) as nvars label='n vars',
                     memlabel
            from             &syslast
          group by   memname;
          describe   table   &syslast;
          select *   from    &syslast;
          quit;
```

log

```
NOTE: SQL table WORK.LIST_MEMNAMES_SASHELP was created like:
   create table WORK.LIST_MEMNAMES_SASHELP
  (LIBNAME  char(8)   label='Library Name',
   MEMNAME  char(32)  label='Library Member Name',
   MEMLABEL char(256) label='Data Set Label',
   NOBS     num       label='Observations in Data Set',
   nvars    num       label='n vars'
```

lst

```
Library
Member     Obs in
Name     Data Set   n vars   Data Set Label
AACOMP       2020        4
AARFM         130        4
ADSMSG        426        6
AFMSG        1090        6
AIR           144        2   airline data (monthly: JAN49-DEC60)...
...
ZIPCODE     41140       21   US Zipcodes; Source: zipcodedownload.com Jan 2017
ZIPMIL        560       21   US Military Zipcodes-lat/long NA-assigned missing
ZTC         18161        6
_CMPIDX_       44       13
```

Options

overview

This is the list of topics in this section:

- syntax, proc options
- list groups
- find word in options
- find group of an option
- list opstart eq startup

Refer to the log saved from program 3, shown on pg. 5,
or use this program to write notes to the log.

```
proc sql; describe table dictionary.options; quit;
```

log

```
NOTE: SQL table DICTIONARY.OPTIONS was created like:
   create table DICTIONARY.OPTIONS
  (optname  char(32)   label='Option Name',
   opttype  char(8)    label='Option type',
   offset   num        label='Offset into option value',
   setting  char(1024) label='Option Setting',
   optdesc  char(160)  label='Option Description',
   level    char(8)    label='Option Location',
   optstart char(8)    label='Option Set',
   group    char(32)   label='Option Group'
```

notes: Table `dictionary.options` has a primary key: column `optname`.

syntax, proc options

```
proc options define value = <optname>; run;
proc options group        = <group>  ; run;
```

list groups

Program 9 lists the option groups and number of options in each.

Program 9 sql-d-options-groups.sas

```
proc sql; select   distinct lowcase(group) as group,
                            count(optname) as count
          from     dictionary.options
          group by group;
          quit;
```

lst

```
group            count
--------------   -----
animation            8
cas                 11
codegen              1
communications      52
dataquality          2
email                9
envdisplay          30
envfiles            35
errorhandling       21
execmodes           21
extfiles             3
graphics             5
help                 8
inputcontrol        15
install              1
languagecontrol     19
listcontrol         13
log_listcontrol     10
logcontrol          22
macro               29
memory               5
meta                13
odsprint            34
pdf                  9
performance         33
sasfiles            36
security            11
sort                11
sql                  9
svg                  9
tk                   2
unknown              1
```

find word in options

Program 10 shows how to find a word in any of the columns in dictionary.options.

Program 10 sql-d-options-find-word.sas

```
%let word = macro;
proc sql; %let word = %lowcase(&word);
          select group, optname, setting, optdesc
            from dictionary.options
           where index(lowcase(optname),"&word")
              or index(lowcase(setting),"&word")
              or index(lowcase(optdesc),"&word");
          quit;
```

lst

```
Option          Option       Option          Option
Group           Name         Setting         Description
MACRO           CMDMAC       NOCMDMAC        Checks window environment commands for command-style macros.
MACRO           IMPLMAC      NOIMPLMAC       Checks for statement-style macros.
MACRO           MACRO        MACRO           Enables the macro facility.
...
MACRO           SASMSTORE                    Specifies the libref of a SAS catalog
                                               for stored compiled SAS macros.
MACRO           SERROR       SERROR          Issues a warning message when a macro variable reference
                                               does not match a macro variable.
MACRO           SYMBOLGEN    NOSYMBOLGEN     Displays the results of resolving macro variable references
                                               in the SAS log.
COMMUNICATIONS  SYSRPUTSYNC  NOSYSRPUTSYNC   Sets the %SYSRPUT macro variables in the client session
                                               when the %SYSRPUT statements are executed.
```

find group of an option

Program 11 shows how to find the group associated with an option in dictionary.options.

Program 11 sql-d-options-find-group.sas

```
%let optname = mprint;
proc sql; select  group
            into :group
            from  dictionary.options
           where  optname eq "%upcase(&optname)";
          quit;
proc options group = &group;
run;
```

log

```
Group=MACRO
 NOCMDMAC     Does not check window environment commands for command-style macros.
 NOIMPLMAC    Does not check for statement-style macros.
 MACRO        Enables the macro facility.
...
  MPRINT            Displays the SAS statements that are generated by macro execution.
...
 SASMSTORE=   Specifies the libref of a SAS catalog for stored compiled SAS macros.
 SERROR       Issues a warning message when a macro variable reference
                 does not match a macro variable.
 NOSYMBOLGEN  Does not display the results of resolving macro variable references
                 in the SAS log.
```

list opstart eq startup

Program 12 shows how to list all the options that are startup-only, i.e: that can only be assigned in a configuration file. See Fehd, “A Configuration File Companion: testing and using environment variables and options; templates for startup-only options initstmt and termstmt” for more information.

Program 12 sql-d-options-optstart.sas

```
%let out = list_opstart_eq_startup;
proc sql;  describe table dictionary.options;
  title3  'select   distinct optstart';
           select   distinct optstart
             from   dictionary.options;
           create   table &out as
           select   group, optname, setting, optdesc
             from   dictionary.options
            where   optstart eq 'startup'
           order by group, optname;
  title3  'select   distinct group';
           select   distinct group
             from   &out;
           quit;
proc print data = &out;
  title3       &out;
           by group;
           id group;
run;
```

lst

```
select distinct optstart
Option Set
----------
anytime
startup

select distinct group
Option Group
------------
communications
envdisplay
envfiles
...
sasfiles
security
tk
```

```
group           optname  setting  optdesc
communications  comaux1           Specifies the first alternate communication access method.
                comaux2           Specifies the second alternate communication access method.
...
group  optname      setting  optdesc
tk     datapagesize  current  Specifies whether the page size for a data set or utility file
                              is compatible with SAS 9.3 processing,
                              or is determined by the current version of SAS.
```

Name collisions in catalogs

overview

Name collisions describes the problem of having two objects with the same name in different locations. In SAS software this can occur when the two entities are in different catalogs.

This is the list of topics in this section:

- describe `dictionary.catalogs`
- formats
  - demo name collision formats
  - finding name collisions in format catalogs
  - catalog delete format
- macro definitions
  - demo name collision macro definitions
  - finding name collisions in macro catalogs
  - catalog delete macro definition

describe dictionary.catalogs

Refer to the log saved from program 3, shown on pg. 5,
or use this program to write notes to the log.

```
proc sql; describe table dictionary.catalogs; quit;
```

log

```
NOTE: SQL table DICTIONARY.CATALOGS was created like:
   create table DICTIONARY.CATALOGS
  (libname  char(8)   label='Library Name',
   memname  char(32)  label='Member Name',
   memtype  char(8)   label='Member Type',
   objname  char(32)  label='Object Name',
   objtype  char(8)   label='Object Type',
   objdesc  char(256) label='Object Description',
```

notes: Table `dictionary.catalogs` has a composite key:
columns `libname + memtype + memname + objname + objtype`.

formats

The format procedure creates lookup tables (associative arrays) in the default catalog: `work.formats`. The catalog name may be assigned in the `library=` option of the format procedure.
There are three possible catalog names:

`library = work.formats`	the default
`library = libref`	memname: formats
`library = libref.memname`	e.g.: same name as dataset

The option `fmtsearch` defines the search list;
the default value is `(work library)`.
The *name collision* problem occurs when same-named format definitions exist in two different catalogs, e.g. `work.formats` and `library.formats`.

demo name collision formats

Program 13 shows the naming collisions problem with the same-named format in both the work (default) and library format catalogs.
Note the example contains a numeric format saved to `work.fmt_num`; this is provided to show that a numeric format has `objtype=FORMAT`, not formatN, when compared to character format where `objtype=FORMATC`.

Program 13 demo-name-collisions-formats.sas

```
*see autoexec for:  ;
*libname library '.';
proc format library  =  work;
     value $sex  'F' = 'female'
                 'M' = 'male';
proc format library  =  work.fmt_num;
     value  sex   0  = 'female'
                  1  = 'male';
proc format library  =  library;
     value $sex  'F' = 'Feminine'
                 'M' = 'Masculine';
options fmtsearch = (work library);         * default list;
%put echo: %sysfunc(getoption(fmtsearch,keyword));
%put echo: %sysfunc(putc(F,$sex.)) %sysfunc(putc(M,$sex.));
options fmtsearch = (library work);
%put echo: %sysfunc(putc(F,$sex.)) %sysfunc(putc(M,$sex.));
```

log

```
NOTE: Format $SEX has been output.
...
NOTE: Format  SEX has been written to WORK.FMT_NUM.
...
NOTE: Format $SEX has been written to LIBRARY.FORMATS.
...
echo: FMTSEARCH=(WORK LIBRARY)
81    %put echo: %sysfunc(putc(F,$sex.)) %sysfunc(putc(M,$sex.));
echo: female male
82    options fmtsearch = (library work);
83    %put echo: %sysfunc(putc(F,$sex.)) %sysfunc(putc(M,$sex.));
echo: Feminine  Masculine
```

finding name collisions in format catalogs

Program 14 shows how sql extracts information from `dictionary.catalogs` about the formats in catalogs named in the `fmtsearch` option.

Program 14 sql-d-catalogs-name-collisions-formats.sas

```
proc sql; create table list_formats as
          select objname, objtype,
                 libname, memname, memtype, objdesc
          from   dictionary.catalogs
          where  libname ne   'SASHELP'    %*excluded;
            and  memtype eq   'CATALOG'
            and  objtype like 'FORMAT%'    %*begins with;
          order  by objname, objtype, libname;
          quit;
proc print data = &syslast;
           by     objname objtype;
           id     objname objtype;
           title 'exist duplicate objname+objtype? '
                 "%sysfunc(getoption(fmtsearch,keyword))";
run;
```

notes: The *libref* `SASHELP` is excluded from the list of formats.

lst

```
exist duplicate objname+objtype? FMTSEARCH=(LIBRARY WORK)
objname   objtype   libname   memname   memtype
-------   -------   -------   -------   -------
SEX       FORMAT    WORK      FMT_NUM   CATALOG

objname   objtype   libname   memname   memtype
SEX       FORMATC   LIBRARY   FORMATS   CATALOG
                    WORK      FORMATS   CATALOG
```

notes: The print procedure with both `by` and `id` statements is used to highlight the duplicate formats.

catalog delete format

Program 15 shows the use of the catalog procedure to remove a format definition.

Program 15 catalog-delete-format.sas

```
 /*name:  catalog-delete-format.sas;
objname   objtype   libname   memname   memtype
-------   -------   -------   -------   -------
SEX       FORMATC   LIBRARY   FORMATS   CATALOG
                    WORK      FORMATS   CATALOG
**** */
proc catalog catalog = work.formats;
             contents;
             delete sex                 %*objname;
                  / entrytype=formatc;  %*objtype;
             quit;
```

log

```
NOTE: Deleting entry SEX.FORMATC in catalog WORK.FORMATS.
```

macro definitions

Macro definitions are defined by the `%macro` and `%mend` statements. The default catalog name is `work.sasmacr`. The *compile+store* macro facility may be used to add a catalog name to the search path of macro definitions. The options enabling the *compile+store* macro facility are:

`mstored`	allow storage
`sasmstore=libref`	location: catalog-name.sasmacr

There is no option to change the search path of macro definitions; the default catalog `work.sasmacr` is always searched first; if the *compile+store* facility is used then the *libref* specified by the option `sasmstore` is searched next.

The *name collision* problem occurs when same-named macro definitions exist in the work and sasmstore catalogs, e.g. `work.sasmacr` and `libref.sasmacr`.

demo name collision macro definition

Program 16 shows the naming collisions problem with the same-named macro definitions in both the work (default) and library catalogs.

Program 16 demo-name-collisions-macro-definitions.sas

```
*see autoexec for:  ;
*filename project '.';
*options  mautosource sasautos = (project sasautos);
*libname  library '.';
options   mstored sasmstore = library;
%macro demo(data=sashelp.air)
       /des = 'demo in library'
        store source;
%put _local_;
%mend  demo;
%demo
%macro demo(data=sashelp.class)
       /des = 'demo in work';
%put _local_;
%mend  demo;
%demo
```

log

```
69   *name: demo-name-collisions-macro-definitions.sas;
70   options mstored sasmstore = library;
71   %macro demo(data=sashelp.air)
72          /des = 'demo in library'
73           store source;
74   %put _local_;
75   %mend  demo;
76   %demo
DEMO DATA sashelp.air
77   %macro demo(data=sashelp.class)
78          /des = 'demo in work';
79   %put _local_;
80   %mend  demo;
81   %demo
DEMO DATA sashelp.class
```

finding name collisions in macro definitions

Program 17 shows the use of `dictionary.catalogs` to find duplicate macro definitions.
Note the fetch of the *libref* from the option `sasmstore`;
! → this *libref* does not have to be `library`.

Program 17 sql-d-catalogs-name-collisions-macro-definitions.sas

```
%let sasmstore = %upcase(%sysfunc(getoption(sasmstore)));
%put echo &=sasmstore;
proc sql; create table list_catalogs_macro_definitions as
          select objname, objtype,
                 libname, memname, memtype, objdesc
            from  dictionary.catalogs
           where  memname like 'SASMAC%'   %*begins with;
             and  memtype eq   'CATALOG'
             and  objtype eq   'MACRO'
             and  libname in  ('WORK', "&sasmstore")
           order  by objname, objtype, libname;
          quit;
data &syslast;
set  &syslast;
by   objname;
if not (    first.objname
        and  last.objname ) then output;
proc print data = &syslast;
           by     objname;
           id     objname;
           title 'exist duplicate objname? in WORK '
                 "&sasmstore";
run;
```

lst

```
exist duplicate objname? in WORK LIBRARY
objname   objtype  libname  memname  memtype  objdesc
-------   -------  -------  -------  -------  ---------------
DEMO      MACRO    LIBRARY  SASMACR  CATALOG  demo in library
          MACRO    WORK     SASMAC1  CATALOG  demo in work
```

notes: Note that the where clause included `memname like 'SASMAC%'`; this listing shows the `memname` in the `libname=work` is `SASMAC1`; your operating system
! → may differ from the documentation which specifies `memname=SASMACR`.

catalog delete macro definition

Program 18 shows the use of the catalog procedure to delete a macro definition.

Program 18 catalog-delete-macro-definition.sas

```
 /*name:  catalog-delete-macro-definition.sas;
objname   libname   memname   memtype   objtype
-------   -------   -------   -------   -------
DEMO      LIBRARY   SASMACR   CATALOG   MACRO
          WORK      SASMAC1   CATALOG   MACRO
**** */
proc catalog catalog = work.sasmac1;
             contents;
             delete demo                %*objname;
                  / entrytype=macro;    %*objtype;
             quit;
```

log

```
NOTE: Deleting entry DEMO.MACRO in catalog WORK.SASMAC1.
```

debrief

Remember that values of most columns are `UPCASE` and that a list of column names must be delimited by commas.

Suggested reading

- Code-Crafters-Inc.com, *Summary of SAS Dictionary Tables and Views*
- Fehd
  - Fehd, “Calculating Cardinality Ratio in Two Steps”
  - Fehd, “Q&A with the macro maven: is sql our lingua franca?”
  - Fehd, “A Configuration File Companion: testing and using environment variables and options; templates for startup-only options initstmt and termstmt”
  - Fehd, “An Autoexec Companion, Allocating Location Names during Startup”
  - Fehd and Carpenter, “List Processing Basics: Creating and Using Lists of Macro Variables”
  - Fehd, “How To Use proc SQL select into for List Processing”
- Hadden, “Proc Catalog, the Wish Book SAS(R) Procedure”
- Hermansen
  - Hermansen, “The Sublime Secrets Of The SAS(R) sqlheads”
  - Hermansen, “Structured Query Language: Logic, Structure, And Syntax”
  - Hermansen, “Ten Good Reasons To Learn SAS(R) Sql”
- Koopmann Jr., “%LibDoc: A library documentation macro”
- McGarry and Hadden, “Ms. Independence (from the SAS(R) Format Library)”
- O’Connor, “Secrets of Macro Quoting Functions — How and Why”: delta quotes
- Whitlock, “Proc SQL — Is it a Required Tool for Good SAS Programming?”

Conclusion

The sql dictionaries provide a unique look into the global symbol table which can be used to research some difficult problems.

Author information

Ronald J. Fehd	Ron.Fehd.macro.maven at gmail dot com
LinkedIn	https://www.linkedin.com/in/ronald-fehd-5125991/
FaceBook	https://www.facebook.com/ron.fehdmacromaven
affiliation	senior maverick, theoretical programmer, Fragile-Free Software Institute
also known as	`macro maven` on SAS-L

References

Code-Crafters-Inc.com (2010). *Summary of SAS Dictionary Tables and Views*. 2 pp.; URL: https://math.wcupa.edu/sciences-mathematics/mathematics/gradstat/documents/DictionaryTablesRefCard.pdf.

Fehd, Ronald J. (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.

— (2018). “An Autoexec Companion, Allocating Location Names during Startup”. In: *SouthEast SAS Users Group Conference Proceedings*. 16 pp.; autocall macros, global symbol table, catrefs, filerefs, librefs, cexist catalogs, exist data set, sasautos. URL: http://www.lexjansen.com/sesug/2018/SESUG2018_Paper-196_Final_PDF.pdf.

— (Sept. 2022a). “A Configuration File Companion: testing and using environment variables and options; templates for startup-only options initstmt and termstmt”. In: *Western Users of SAS Software Annual Conference Proceedings*. 18 pp.; URL: https://www.lexjansen.com/wuss/2022/WUSS-2022-Paper-64.pdf.

— (Sept. 2022b). “Calculating Cardinality Ratio in Two Steps”. In: *Western Users of SAS Software Annual Conference Proceedings*. 23 pp.; URL: https://www.lexjansen.com/wuss/2022/WUSS-2022-Paper-69.pdf.

— (Oct. 2022c). “Q&A with the macro maven: is sql our lingua franca?” In: *SouthEast SAS Users Group Conference Proceedings*. 9 pp.; URL: https://www.lexjansen.com/sesug/2022/SESUG2022_Paper_124_Final_PDF.pdf.

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

Hadden, Louise (Sept. 2015). “Proc Catalog, the Wish Book SAS(R) Procedure”. In: *SouthEast SAS Users Group Conference Proceedings*. 6 pp.; URL: https://analytics.ncsu.edu/sesug/2015/CC-58.pdf.

Hermansen, Sigurd (Oct. 1996). “Ten Good Reasons To Learn SAS(R) Sql”. In: *SouthEast SAS Users Group Conference Proceedings*. 5 pp.; URL: https://www.lexjansen.com/sesug/1996/SESUG96017.pdf.

— (Sept. 2002). “Structured Query Language: Logic, Structure, And Syntax”. In: *SouthEast SAS Users Group Conference Proceedings*. 10 pp.; URL: https://analytics.ncsu.edu/sesug/2002/TU24.pdf.

— (Sept. 2003). “The Sublime Secrets Of The SAS(R) sqlheads”. In: *SouthEast SAS Users Group Conference Proceedings*. 6 pp.; URL: https://analytics.ncsu.edu/sesug/2003/SE05-Hermansen.pdf.

Koopmann Jr., Richard (2009). “%LibDoc: A library documentation macro”. In: *SAS Global Forum Annual Conference Proceedings*. Coders Corner, 6 pp.; URL: https://support.sas.com/resources/papers/proceedings09/066-2009.pdf.

McGarry, Nancy and Louise Hadden (Oct. 2019). “Ms. Independence (from the SAS(R) Format Library)”. In: *SouthEast SAS Users Group Conference Proceedings*. 6 pp.; URL: https://www.lexjansen.com/sesug/2019/SESUG2019_Paper-196_Final_PDF.pdf.

O’Connor, Susan M. (Oct. 1999). “Secrets of Macro Quoting Functions — How and Why”. In: *NorthEast SAS Users Group Conference Proceedings*. 12 pp.; URL: https://www.lexjansen.com/nesug/nesug99/bt/bt185.pdf.

Whitlock, Ian (Aug. 2001). “Proc SQL — Is it a Required Tool for Good SAS Programming?” In: *SouthEast SAS Users Group Conference Proceedings*. 6 pp.; URL: https://analytics.ncsu.edu/sesug/2001/P-829.pdf.