Using SQL Dictionaries to Research the Global Symbol Table
Ronald J. Fehd, senior maverick, theoretical programmer,
Fragile-Free Software Institute

Abstract
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.

In this paper:

<table>
<thead>
<tr>
<th>How many sql dictionaries?</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dataset, variables: columns in tables</td>
<td>6</td>
</tr>
<tr>
<td>columns: variable names</td>
<td>7</td>
</tr>
<tr>
<td>tables: dataset names</td>
<td>8</td>
</tr>
<tr>
<td>compare to proc contents</td>
<td>10</td>
</tr>
<tr>
<td>Options</td>
<td>12</td>
</tr>
<tr>
<td>find word in options</td>
<td>14</td>
</tr>
<tr>
<td>find group of an option</td>
<td>14</td>
</tr>
<tr>
<td>list opstart eq startup</td>
<td>15</td>
</tr>
<tr>
<td>Name collisions in catalogs</td>
<td>16</td>
</tr>
<tr>
<td>formats</td>
<td>16</td>
</tr>
<tr>
<td>macro definitions</td>
<td>19</td>
</tr>
<tr>
<td>Suggested reading</td>
<td>21</td>
</tr>
<tr>
<td>Conclusion</td>
<td>22</td>
</tr>
</tbody>
</table>
Introduction

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

Program 1  sql-describe-libref-memname.sas

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

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>
```

Note: in this table name is a column (sql) or variable (dataset) name.
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 lowercase 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:
\texttt{describe table dictionary.<memname>;}
for each of the sql dictionaries.

\textbf{Program 3} \texttt{sql-d-dictionaries-memname-describe-each-1.sas}, \texttt{select into :list}
\texttt{proc sql; select distinct catt('describe table dictionary.'<memname>',;')}
\texttt{into :list separated by ' '}
\texttt{from dictionary.dictionaries;}
\texttt{&list}
\texttt{quit;}
\texttt{%symdel list;}

notes: \texttt{&list} submits the statements in the macro variable list
\texttt{%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:
\texttt{prefix = describe table dictionary.} and \texttt{suffix = semicolon;}.
Program 3 is \textbf{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: \texttt{select '...' length = ??? into :list}
maximum length: of the macro variable list is limited to
\texttt{(2**16)-2 = 65,534;}
\texttt{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()  
 Either of programs 3 or 4 produces this output:

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'
)
```

<table>
<thead>
<tr>
<th>List</th>
<th>Library</th>
<th>Member</th>
<th>Column Name</th>
<th>Column Type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SASHHELP</td>
<td>CLASS</td>
<td>1 Name</td>
<td>char</td>
</tr>
<tr>
<td></td>
<td>SASHHELP</td>
<td>CLASS</td>
<td>2 Sex</td>
<td>char</td>
</tr>
<tr>
<td></td>
<td>SASHHELP</td>
<td>CLASS</td>
<td>3 Age</td>
<td>num</td>
</tr>
<tr>
<td></td>
<td>SASHHELP</td>
<td>CLASS</td>
<td>4 Height</td>
<td>num</td>
</tr>
<tr>
<td></td>
<td>SASHHELP</td>
<td>CLASS</td>
<td>5 Weight</td>
<td>num</td>
</tr>
</tbody>
</table>

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".

<table>
<thead>
<tr>
<th>class</th>
<th>num</th>
<th>cr_type</th>
<th>ratio</th>
<th>levels</th>
<th>name</th>
<th>type</th>
<th>length</th>
<th>etc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>class</td>
<td>1</td>
<td>unique</td>
<td>1</td>
<td>19</td>
<td>Name</td>
<td>c</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>class</td>
<td>2</td>
<td>few</td>
<td>0.10526</td>
<td>2</td>
<td>Sex</td>
<td>c</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>class</td>
<td>3</td>
<td>few</td>
<td>0.31579</td>
<td>6</td>
<td>Age</td>
<td>n</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>class</td>
<td>4</td>
<td>many</td>
<td>0.89474</td>
<td>17</td>
<td>Height</td>
<td>n</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>class</td>
<td>5</td>
<td>many</td>
<td>0.78947</td>
<td>15</td>
<td>Weight</td>
<td>n</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

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.

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

LOG

NOTE: SQL table DICTIONARY.TABLES was created like:

```sas
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.
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'

<table>
<thead>
<tr>
<th>Library</th>
<th>Member</th>
<th>Number of Physical Observations</th>
<th>Number of Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>SASHELP</td>
<td>ACOMP</td>
<td>2020</td>
<td>4</td>
</tr>
<tr>
<td>SASHELP</td>
<td>AARFM</td>
<td>130</td>
<td>4</td>
</tr>
<tr>
<td>SASHELP</td>
<td>ADSMSG</td>
<td>426</td>
<td>6</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SASHELP</td>
<td>ZIPCODE</td>
<td>41140</td>
<td>21</td>
</tr>
<tr>
<td>SASHELP</td>
<td>ZIMIL</td>
<td>560</td>
<td>21</td>
</tr>
<tr>
<td>SASHELP</td>
<td>ZTC</td>
<td>18161</td>
<td>6</td>
</tr>
<tr>
<td>SASHELP</td>
<td><em>CMPIDX</em></td>
<td>44</td>
<td>13</td>
</tr>
</tbody>
</table>
compare to proc contents

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

**Program 7  make-list-names-contents.sas**

```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**

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

Program 8 make-list-memnames-contents.sas

```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'
```

List

```
<table>
<thead>
<tr>
<th>Library Member Name</th>
<th>Obs in Data Set</th>
<th>n vars</th>
<th>Data Set Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>AACOMP</td>
<td>2020</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>AARFM</td>
<td>130</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>ADSMSG</td>
<td>426</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>APMMSG</td>
<td>1096</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>AIR</td>
<td>144</td>
<td>2</td>
<td>airline data (monthly: JAN49-DEC60)...</td>
</tr>
<tr>
<td>ZIPCODE</td>
<td>41140</td>
<td>21</td>
<td>US Zipcodes; Source: zipcodedownload.com Jan 2017</td>
</tr>
<tr>
<td>ZIPMIL</td>
<td>560</td>
<td>21</td>
<td>US Military Zipcodes-lat/long NA-assigned missing</td>
</tr>
<tr>
<td>ZTC</td>
<td>18161</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>
```

_[CMPIDX]_ 44 13
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;

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

```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;
```

<table>
<thead>
<tr>
<th>Option</th>
<th>Option</th>
<th>Option</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group</td>
<td>Name</td>
<td>Setting</td>
<td>Description</td>
</tr>
<tr>
<td>MACRO</td>
<td>CMDMAC</td>
<td>NOCMDMAC</td>
<td>Checks window environment commands for command-style macros.</td>
</tr>
<tr>
<td>MACRO</td>
<td>IMPLMAC</td>
<td>NOIMPLMAC</td>
<td>Checks for statement-style macros.</td>
</tr>
<tr>
<td>MACRO</td>
<td>MACRO</td>
<td>MACRO</td>
<td>Enables the macro facility.</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MACRO</td>
<td>SASMSTORE</td>
<td></td>
<td>Specifies the libref of a SAS catalog for stored compiled SAS macros.</td>
</tr>
<tr>
<td>MACRO</td>
<td>SERROR</td>
<td>SERROR</td>
<td>Issues a warning message when a macro variable reference does not match a macro variable.</td>
</tr>
<tr>
<td>MACRO</td>
<td>SYMBOLGEN</td>
<td>NSYMBOLGEN</td>
<td>Displays the results of resolving macro variable references in the SAS log.</td>
</tr>
<tr>
<td>COMMUNICATIONS</td>
<td>SYSPRUTSYNC</td>
<td>NSYSPRUTSYNC</td>
<td>Sets the %SYSPUT macro variables in the client session when the %SYSPUT statements are executed.</td>
</tr>
</tbody>
</table>

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

```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

<table>
<thead>
<tr>
<th>Group</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MACRO</td>
<td>Does not check window environment commands for command-style macros.</td>
</tr>
<tr>
<td>NOCMDMAC</td>
<td>Does not check for statement-style macros.</td>
</tr>
<tr>
<td>NOIMPLMAC</td>
<td></td>
</tr>
<tr>
<td>MACRO</td>
<td>Enables the macro facility.</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>MPRINT</td>
<td>Displays the SAS statements that are generated by macro execution.</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>SASMSTORE</td>
<td>Specifies the libref of a SAS catalog for stored compiled SAS macros.</td>
</tr>
<tr>
<td>SERROR</td>
<td>Issues a warning message when a macro variable reference does not match a macro variable.</td>
</tr>
<tr>
<td>NSYMBOLGEN</td>
<td>Does not display the results of resolving macro variable references in the SAS log.</td>
</tr>
</tbody>
</table>
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

```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;
```

### List

<table>
<thead>
<tr>
<th>Option Set</th>
<th>Option Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>anytime</td>
<td>communications</td>
</tr>
<tr>
<td>startup</td>
<td>envdisplay</td>
</tr>
<tr>
<td></td>
<td>envfiles</td>
</tr>
<tr>
<td></td>
<td>sasfiles</td>
</tr>
<tr>
<td></td>
<td>security</td>
</tr>
<tr>
<td></td>
<td>tk</td>
</tr>
</tbody>
</table>

- **group** optname setting optdesc
  - Specifies the first alternate communication access method.
  - Specifies the second alternate communication access method.
- **group** optname setting optdesc
  - 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.

```sql
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',
  )

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.
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
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.

<table>
<thead>
<tr>
<th>lst</th>
<th>exist duplicate objname+objtype? FMTSEARCH=(LIBRARY WORK)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>objname</td>
</tr>
<tr>
<td>-----</td>
<td>----------</td>
</tr>
<tr>
<td>SEX</td>
<td>FORMATC</td>
</tr>
<tr>
<td>WORK</td>
<td>FORMATS</td>
</tr>
</tbody>
</table>

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

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 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
- sasmstore=libref

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.

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;
%macro demo(data=sashelp.class)
    /des = 'demo in work';
    %put _local_;%mend 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_;%mend demo;
75 %macro demo(data=sashelp.class)
76 /des = 'demo in work';
77 %put _local_;%mend demo;
78 %macro demo(data=sashelp.air)
79 /des = 'demo in library'
80 store source;
81 %put _local_;%mend demo;
82 %macro demo(data=sashelp.class)
83 /des = 'demo in work';
84 %put _local_;%mend demo;
```
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`

```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%'
         and memtype eq 'CATALOG'
         and objtype eq 'MACRO'
    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; runaway;
```

### List

<table>
<thead>
<tr>
<th>objname</th>
<th>objtype</th>
<th>libname</th>
<th>memname</th>
<th>memtype</th>
<th>objdesc</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEMO</td>
<td>MACRO</td>
<td>WORK</td>
<td>SASMACR</td>
<td>CATALOG</td>
<td>demo in work</td>
</tr>
</tbody>
</table>

### 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`.  

20
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 UPPCASE 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 inststmt 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) sqheaders"
  – 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
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

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. © indicates USA registration.
References


