How to retrieve SAS License Information, if you have a local installation of SAS

```
%macro getInfo;
  options nosyntaxcheck ;
  *LIBNAME _ALL_ LIST;
  %put Site Number: &syssite ;
  %put Host OS: &sysscp; %put &sysscpl;
  %put Hostname: &systcpiphostname ;
  %put Process: &sysprocessname ;
  %put SAS Version: &sysvlong ;
  %let sasroot=%sysget(SASROOT) ;
  %put SASROOT: &sasroot ;
  %put USER: &sysuserid ;
  %put Bitness: &SYSADDRBITS ;
  %put Username: %SYSGET(USERNAME) ;
  %put Random: %SYSGET(SAS_NO_RANDOM_ACCESS) ;
  %put This job started on &sysdate at &systime;
%mend ;
%getInfo ;
```

ProcImport.md

Method 1: Importing an Excel Spreadsheet into a SAS data set

```
options nodate nonumber nodate;
  PROC IMPORT DATAFILE= "C:\SASCourse\Week2\class.xlsx"
     dbms=xlsx REPLACE OUT= work.class_x;
     SHEET="Sheet1";
     GETNAMES=YES;
 RUN;
Title;
 proc print data=work.class_x (obs=5);
 run;
```

Method 2: Using LIBNAME XLSX to read Excel files

The LIBNAME statement references the whole Excel file.
The SET statement uses the Excel sheet as an input data file for this DATA step.

```
options validvarname=any;
libname XL XLSX 'C:\SASCourse\Week2\class.xlsx';
data work.class;
  set XL.Sheet1;
run;
libname XL CLEAR;
proc print data=work.class (obs=5);
run;
```

Another Example

```
%let Path = C:\SASCourse\Week2;
options validvarname=any;
Libname XL XLSX "&Path\MEPS_MCCC.xlsx";
Libname new "C:\SASCourse\Week2";
data new.MCCC_Data(keep=CCSR_Code MCCC);
  set XL.sheet1;
run;
libname XL CLEAR;
```

ProcExport.md

The following SAS program converts the SAS data set to a CSV file.

```
dm "log; clear; output; clear; odsresults; clear;";
Proc export data=sashelp.demographics
       (keep=region name pop
        rename=(region=%sysfunc(lowcase(region))
                name=%sysfunc(lowcase(name))
                          pop=%sysfunc(lowcase(pop))
               )
       )
    outfile="c:\data\sashelp_demographics.csv"
    dbms=CSV  replace ;
run;
```

The following SAS program converts the SAS data set to an Excel spreadsheet.

```
/* format the current date as "YYYY-MM-DD" */
%let current_date = %sysfunc(today());
%let formatted_date = %sysfunc(putn(&current_date., yymmdd10.));
proc export
        data=sashelp.class
        dbms=xlsx
        outfile="C:\Data\Class_&formatted_date..xlsx"
        replace;
run;
```

The following code converts the SAS data set to a Stata data set.

```
proc export data=sashelSp.class replace
         outfile= "C:\Data\class.dta";
run;
```

HowToCopySASOutputExcel.md

Write multiple SAS outputs to various Excel sheets

The 'PROC' option creates a new Excel worksheet for each SAS procedure.

```
ods excel file = "c:\SASCourse\Week7\class_classfit.xlsx"
     options(sheet_name="class" embedded_titles="yes" sheet_interval='PROC');
  title 'PROC CONTENTS - SASHELP.CLASS File';
 proc contents data=SASHELP.CLASS position;
 ods select variables;
 run;
 ods excel options(sheet_name="classfit" embedded_titles="yes"
sheet_interval='PROC');
 title 'PROC CONTENTS - SASHELP.CLASSFIT File';
 proc contents data=SASHELP.CLASSFIT position;
 ods select variables;
 run;
 ods excel close;
```

Write multiple SAS outputs to the same Excel sheet.

The 'NONE' option creates all the output on the same page (Excel sheet).

```
ods excel file = "c:\SASCourse\Week7\class_classfit_x.xlsx"
               options(embedded_titles="yes" sheet_interval="none");
  title 'PROC CONTENTS - SASHELP.CLASS File';
 proc contents data=SASHELP.CLASS position;
 ods select variables;
 run;
 ods excel options(embedded_titles="yes" sheet_interval="none");
 title 'PROC CONTENTS - SASHELP.CLASSFIT File';
 proc contents data=SASHELP.CLASSFIT position;
 ods select variables;
 run;
 ods excel close;
```

Write two SAS outputs per Excel spreadsheet.

Trick: Suspend a destination temporarily after writing the first two outputs to the first spreadsheet before writing the second set of output to a new Excel spreadsheet.

```
ods listing close; /* Closing the default listing output */
ods excel file="C:\Data\TwoOutputPerShhet.xlsx"
      options(embedded_titles='yes' embedded_footnotes='yes'
      title_footnote_width="80"); /* Open an Excel destination */

/* Create the first sheet named "Heart_Cars" and prints two tables */
ods excel options(sheet_name="Heart_Cars" sheet_interval="none");
title "Table 1: SASHELP.HEART data set's variable attributes";
proc print data=sashelp.vcolumn label;
 var name type format length label;
 where libname='SASHELP' and memname='HEART';
run;

title "Table 2: SASHELP.CARS data set's variable attributes";
proc print data=sashelp.vcolumn label;
 var name type format length label;
 where libname='SASHELP' and memname='CARS';
run;

/* Suspend a destination temporarily and move to a new Excel spreadsheet */
ods select none;
ods excel options(sheet_interval="proc");
proc print data=sashelp.class; run;
ods select all;

/* Create another sheet "Demographics_IRIS" and prints two more tables  */
title "Table 3: SASHELP.DEMOGRAPHICS data set's variable attributes";
ods excel options(sheet_name="Demographics_IRIS" sheet_interval="none");
proc print data=sashelp.vcolumn label;
 var name type format length label;
 where libname='SASHELP' and memname='DEMOGRAPHICS';
run;

title "Table 4: SASHELP.IRIS data set's variable attributes";
proc print data=sashelp.vcolumn label;
 var name type format length label;
 where libname='SASHELP' and memname='IRIS';
run;
ods excel close;
title;
ods listing;
```

This program creates a report on the SASHELP datasets using PROC SQL and saves it to an Excel file. The report includes the dataset name, number of observations, variables, creation date, and creation time.

```
ods excel file = 'C:\Data\SASHELP_Datasets_10_14_2020.xlsx'
   options (embedded_titles='on'  sheet_name='List');
title "Listing of SASHELP Data Sets";
proc sql number;
  select memname,nobs format=comma10.
         ,nvar format=comma7.
         ,DATEPART(crdate) format date9. as Date_created label='Creation
Date'
         ,TIMEPART(crdate) format timeampm. as Time_created label='Creation
Time'
       from dictionary.tables
       where libname = 'SASHELP'
        and memtype = 'DATA';
     quit;

  ods excel close;
  ods listing;
```

The RETAIN statement and the assignment statement with the + operator

In this SAS program, the `RETAIN` statement initializes the variable `SUM_X` to 0, and the assignment statement with the `+` operator adds the value of `X` to `SUM_X` for each observation.

```
data sum_example;
  input x 1-5;
  retain sum_x 0;
  sum_x = sum_x + x;
cards;
10
20
30
40
50
;
run;
```

This code uses a clever trick in SAS to calculate the cumulative sum without using the `RETAIN` statement or the `SUM` function. `SUM_X + X;` is called the SUM statement. The `datalines` statement inputs the data, and the `PROC PRINT` statement prints the resulting dataset.

```
data sum_example2;
  input x ;
sum_x + x;
datalines;
10
20
30
40
50
;
run;
proc print data=sum_example2;
run;
```

In SAS, the PDV (Program Data Vector) is the area of memory where SAS stores the current observation being processed.

When you use the SUM statement in SAS, it automatically retains the previous value of the variable and adds the new value to it, without the need for an explicit RETAIN statement.

This SAS program sorts the `SASHELP.CARS dataset by the MAKE variable. A DATA step` creates a new CARS dataset with an additional variable, COUNT, that enumerates the observations within each `MAKE` group. Note that `COUNT + 1` is called the SUM statement.

```
proc sort data = sashelp.cars; by make; run;
data cars;
  set cars;
  count + 1;
  by make;
  if first.make then count = 1;
  if last.make;
run;
```

Here's a breakdown of the code:

Proc Sort Statement

`proc sort data = sashelp.cars; by make; run;`

- This statement sorts the SASHELP.CARS dataset by the MAKE variable in ascending order.

Data Step

data cars;

- This statement starts a new data step to create a new dataset called CARS.

Set Statement

set sashelp.cars;

- This statement reads the sorted CARS dataset into the data step.

Count Variable

count + 1;

- This statement creates a new variable COUNT and initializes it to 1. The + 1 increments the COUNT variable by 1 for each observation within each MAKE group.

By Statement

by make;

- This statement specifies that the data step should process the observations within each MAKE group.
-

If First.Make Statement

if first.make then count = 1;

- This statement resets the COUNT variable to 1 when encountering a new MAKE group.
-

If Last.Make Statement

if last.make;

- This statement outputs the last observation of each MAKE group to the new CARS dataset. The resulting CARS dataset will have the same variables as the original SASHELP.CARS dataset, plus the additional COUNT variable, enumerates each MAKE group's observations.

This SAS program sorts the SASHELP.CARS dataset by the MAKE variable and then creates a new dataset called COUNTS with the count of observations for each MAKE group.

```
proc sort data=sashelp.cars out=cars; by make; run;
data counts (keep= make model type count_of_make);
   set cars; by make;
   if first.make then count_of_make=0;
    count_of_make+1;
   if last.make;
  run;
```

Here's a breakdown of the code:

Proc Sort Statement
`proc sort data=sashelp.cars out=cars; by make; run;`

- This statement sorts the SASHELP.CARS dataset by the MAKE variable and outputs the sorted dataset to a new CARS dataset.

Data Step
`data counts (keep= make model type count_of_make);`

- This statement starts a new data step to create a new dataset called COUNTS and specifies the variables to keep in the output dataset.

Set Statement
`set cars;`

- This statement reads the sorted CARS dataset into the data step.

By Statement
`by make;`

- This statement specifies that the data step should process the observations within each MAKE group.

If First.Make Statement
`if first.make then count_of_make=0;`

- This statement resets the COUNT_OF_MAKE variable to 0 when encountering a new MAKE group.

Count_of_make Statement
`count_of_make+1;`

- This statement increments the COUNT_OF_MAKE variable by 1 for each observation within the MAKE group.

If Last.Make Statement

```
if last.make;
```

- This statement outputs the last observation of each `MAKE` group to the new `COUNTS` dataset.

The resulting `COUNTS` dataset will contain the following variables:

- `MAKE`
- `MODEL`
- `TYPE`
- `COUNT_OF_MAKE` (the count of observations for each `MAKE` group)

Note that the `KEEP` statement in the `DATA` step specifies the variables to include in the output dataset. In this case, the MODEL and TYPE variables are included, but they will have the same value for all observations within each MAKE group since the data step only outputs the last observation of each group. You can modify the KEEP statement accordingly if you only want to keep the MAKE and COUNT_OF_MAKE variables.

```
import pandas as pd

# Load the dataset (assuming it's a CSV file)
data = pd.read_csv('sashelp_cars.csv')

# Sort the data by 'make'
data = data.sort_values(by='make')

# Group by 'make' and count the number of observations
counts = data.groupby('make').size().reset_index(name='count_of_make')

# Keep only the 'make', 'model', and 'count_of_make' columns
counts = counts.merge(data[['make', 'model']].drop_duplicates(), on='make')

# Print the result
print(counts)
```

Note that this script assumes `SASHELP.CARS` dataset is stored in a CSV file named `sashelp_cars.csv`. You may need to modify the script if your data is stored in a different format or location.

Also, this script uses the `groupby` method to count the number of observations for each `make` group, which is equivalent to the `BY` statement and `COUNT_OF_MAKE` variable in the SAS program. The `merge` method adds the `model` column to the result since the `groupby` method only returns the grouping column and the count.

- `data.groupby('make')`: This groups the data by the 'make' column, creating a GroupBy object. This is similar to the `BY` statement in SAS.

- `.size()`: This method returns the number of observations in each group. It's equivalent to the `COUNT_OF_MAKE` variable in the SAS program.
- `.reset_index(name='count_of_make')`: This method resets the index of the resulting DataFrame and renames the count column to 'count_of_make'.

  So, the entire line of code says: "Group the data by 'make', count the number of observations in each group, and create a new DataFrame with the 'make' column and the count column named 'count_of_make'".

Based on SASHELP.DEMOGRAPHICS, the SAS program calculates:

- the sum and percentage of the total population for each region
- the total population and the percentage (which will always be 100%) for all regions combined.

```
proc format;
value $regionfmt
    'AFR' = 'Africa'
    'AMR' = 'Americas'
    'EUR' = 'Europe'
    'EMR'  ='Eastern Mediterranean'
    'SEAR' = 'South-East Asia'
    'WPR' = 'Western Pacific';
invalue order
    'AFR' = 1
    'AMR' = 2
    'EUR' = 3
    'EMR'  = 4
    'SEAR' = 5
    'WPR' = 6
    other = 7;
run;


proc sql;
create table want1 as(
select region format= $regionfmt.,
      sum(pop) as sum_pop format=comma14.,
         sum(pop)/ (select sum(pop) from  sashelp.demographics)*100 as
percent_pop format=8.1
    from
        sashelp.demographics
    group by region

                      )
  union
 select 'Total',
         sum(pop) as sum_pop format=comma14.,
         sum(pop)*100/sum(pop) as percent_pop format=8.1
   from sashelp.demographics;
  select *
   from want1
   order by input(region, order.);
quit;
```

This SQL code creates a table named want1 using data from the sashelp.demographics dataset. Here's a breakdown of what the code does:

1. It starts a PROC SQL procedure.
2. It creates a table want1 that combines two SELECT statements using a UNION.
3. The first SELECT statement:
    - Select the region, sum of population, and percentage of the total population for each region.
    - Groups the results by region.
4. The second SELECT statement:
    - Calculates the total sum of population and percentage (which will always be 100%) for all regions combined.
    - Uses 'Total' as the region name for this row.
5. Finally, it selects all columns from the want1 table and orders the results based on the region column, using a custom order defined by the order. format.

How To Add Total to Table

```
/*Create the total row*/
proc means noprint data=sashelp.cars ;
  output out=summary(keep=mpg_city) sum=mpg_city;
run;

/*Create the necessary table using your proc sql*/
proc sql;
        create table test as
                select make, count(make) as count
                from sashelp.cars
                group by make
                order by make;
quit;

/*Combine the tables*/
data want ;
  set test summary(in=in2); /*concatenate the tables. In= option creates a
temp variable called in2*/
  if in2=1 then Make='TOTAL'; /*In2 will =1 when SAS brings in the row from
summary. This will allow us to change the column name we want to "Total"*/
run;
```

Alternative code

```
proc sql;
    select make, count(make) as count
    from sashelp.cars
    group by make
    UNION
    select 'Total' as make,
           count(*) as count format=comma14.
    from sashelp.cars
    order by case when make = 'Total' then 1 else 0 end,
             make;
quit;
```

Extended Example

```
proc sql;
    select make,
           count(make) as count,
           (count(make) / (select count(*)
            from sashelp.cars)) * 100 as percentage format=5.2
    from sashelp.cars
    group by make

    UNION

    select 'Total' as make,
           count(*) as count,
           100.00 as percentage format=5.2
    from sashelp.cars

    order by case when make = 'Total' then 1 else 0 end,
             make;
quit;
```

This query does the following:

1. In the first SELECT:
    - It counts the occurrences of each make.
    - It calculates the percentage by dividing the count of each make by the total number of cars (obtained from the subquery) and multiplying by 100.
    - The percentage is formatted to display with 2 decimal places.
2. In the UNION part:
    - It adds a 'Total' row with the total count of cars.
    - For the 'Total' row, the percentage is set to 100.00.
3. The ORDER BY clause remains the same, keeping the 'Total' row at the end.

This query will give you:

- The make of each car
- The count of each make
- The percentage that each make represents of the total
- A 'Total' row at the end with the total count and 100% percentage

The percentage column will show what percent each make represents of the total number of cars in the dataset.

Alternative code

This code produces a report summarizing the cars by make, showing the count and percentage for each make, with a total row at the end.

```
proc report data = sashelp.cars nowd;
 column make n pctn;
 define make /group id order = internal;
 define n / format =8. "N";
 define pctn / "Percentage" format =percent7.1;
 rbreak after /summarize style=[font_style=italic];
 compute after ;
          make='Total';
endcomp;
run;
```

ProcReport.md

```
DATA work.HC_spend1;
 INFILE DATALINES DLM=',' MISSOVER;
 INPUT service_type: & $ 32. amount_in_B;
 DATALINES;
 Hospital Care, 882.3
 Physician and Clinical Services, 556.0
 Prescription Drugs, 263.3
 Nursing Care, 151.5
 Other Health Care, 138.2
 Dental Services, 110.9
 Home Health Care, 77.8
 Other Medical Products, 95.0
 Other Professional Services, 78.4
 ;
proc report data=work.HC_spend1 nowd SPLIT='*';
column service_type amount_in_B Percent;
define service_type / display  "Type of Services";
define amount_in_B / analysis SUM FORMAT=Dollar8.1 "Expenses (in Billions)"
Width=15;
define Percent / computed FORMAT=Percent8.1;

*Create Temporary Variables;
compute before;
 denom=amount_in_b.sum;
endcomp;

* Create a New Variable;
 compute Percent;
   Percent= amount_in_B.sum / denom;
 endcomp;

rbreak after / summarize UL OL;
Title1 "Health Care Expenditures by Type of Services, United States, 2012";
Title2 '  ';
run;
```

```
proc report data=sashelp.demographics  headline headskip;
  column region pop pop=sum pop=pct;
  define region / group format=$regionfmt. ;
  define pop / analysis 'N'  n;
  define sum / analysis 'Sum'  format=comma14.;
  define pct / analysis 'Percent of Total' pctsum format=percent8.1;

  compute after;
      region = 'Total';
  endcomp;
 rbreak after / skip summarize ;
run;
```

```
proc report data=sashelp.class nowd;
  column name age height weight bmi;
  define name / group;
  define age / group noprint;  /* Age used for grouping but not displayed */
  define height / analysis mean;
  define weight / analysis mean;
  define bmi / computed format=5.1;
  compute bmi;
    bmi = (weight.mean / (height.mean/100)**2) * 703;
  endcomp;
run;
```

```
proc report data=sashelp.cars nowd;
   column make type invoice invper msrp msrpper;
   define make / group;
   define type / group;
   define invper / computed format=percent8.1 'Invoice/Percent';
   define msrpper / computed format=percent8.1 'MSRP/Percent';

   /* Put the total sum of Type within Make into a DATA step variable */
   compute before make;
      invden = invoice.sum;
      msrpden = msrp.sum;
   endcomp;

/* Compute the percents */
   compute invper;
      if invden > 0 then invper = invoice.sum / invden;
   endcomp;
   compute msrpper;
      if msrpden > 0 then msrpper = msrp.sum / msrpden;
   endcomp;

    compute after make;
     type = 'Total';
   endcomp;
  break after make / summarize suppress skip;
 run;
```

ProcDatasets.md

Remove all labels and formats using the MODIFY statement and the ATTRIB option within PROC DATASETS.

```
proc datasets lib=work memtype=data;
     modify a;
     attrib _all_ format=;
contents data=a;
run;
quit;
```

Delete all SAS data sets from multiple libraries using the KILL option with PROC DATASETS

```
libname new ('C:\Data\Zip_folder' 'C:\Data\Xpt_folder'
             'C:\Data\Cpt_folder' 'C:\Data\SDS_folder');
proc datasets nolist library=new kill;
quit;
```

Delete selected SAS data sets from the WORK library

```
proc datasets nolist lib=work;
 delete a1 a2 a3 ;
quit;
```

HowToDirList.md

Topic: How to create folders dynamically (Various)

```
%LET RootPath=c:\mydata;
%LET Folder=%SYSFUNC(DCREATE(MyNewFolder,&RootPath));
%LET RootPath=C:\;
Data _Null_;
SubDIR='TeachingSAS';
FolderName=DCREATE(SubDIR,"&RootPath");
Put FolderName=;
Run;
%LET Folder=%SYSFUNC(DCREATE(MacroFolder,&RootPath));
%PUT Folder=&Folder;

/* This macro creates multiple folders */

%LET RootPath=c:\SASCourse;
%MACRO CF (start, stop);
  %DO num= &start %to &stop;
     %LET Folder1=%SYSFUNC(DCREATE(Week&num,&RootPath));
  %END;
%MEND CF;
%CF(1,15)
%let TargetPath=c:\data\temp\Folder;
FILENAME FMyRep "&TargetPath";
%LET rc=%SYSFUNC(FDELETE(FMyRep));
FILENAME FMyRep CLEAR;
```

Turning external files into SAS® data sets: common problems and their solutions

The TRUNCOVER option indicates that the program should not flow over to the following line to look for missing values not found on the current line being read.

```
Filename filelist pipe "dir /b /s c:\SASCourse\Week1\*.sas";
   Data Listfiles;
    Length very_last_word $8;
     Infile filelist truncover;
     Input filename $100.;
     very_last_word=scan(filename, -1);

     * Last words from the reverse direction delimited by a slash;
     File_name=substr(scan(filename, -1, '\'),1);
     Run;
proc sort data=Listfiles; by File_name; run;
proc print data=Listfiles;
var  File_name;
where very_last_word eq 'sas';
run;
```

Additional SAS code examples

```
Filename filelist pipe "dir /b /s c:\temp\*.sas";
   Data _null_;
     Infile filelist truncover;
     Input filename $100.;
     Put filename=;
 Run;
```

```
filename DIRLIST pipe 'dir /b /s c:\Data\*.sas';
data dirlist;
length filename $200;
infile dirlist length=reclen;
input filename $varying200. reclen;
run;

proc print data=dirlist;
var file_name;
where scan(filename, -1, '.')='sas';
run;
```

List file names using R

```
* List file names from a folder - Variant 1;
PROC IML;
SUBMIT / R;
setwd ("C:/SASCourse/Week4")
list.files(pattern="^", full.names = TRUE, ignore.case = TRUE)
ENDSUBMIT;
QUIT;
```

```
* List file names from a folder - Variant 2;
PROC IML;
SUBMIT / R;
setwd ("C:/SASCourse/Week4")
list.files(pattern="SAS", full.names = TRUE, ignore.case = TRUE)
list.files(pattern="txt", full.names = TRUE, ignore.case = TRUE)
ENDSUBMIT;
QUIT;
```

```
* List file names from a folder - Variant 3;
PROC IML;
SUBMIT / R;
list.files (path = "C:/SASCourse/Week4")
ENDSUBMIT;
QUIT;
```

Search for SAS files in a specified directory and its subdirectories and print the full path of each found file using Python

```
import os
path = r'C:\Misc'
files = []
for r, d, f in os.walk(path):
    for file in f:
        if '.sas' in file:
            files.append(os.path.join(r, file))

for f in files:
    print(f)
```

- Import and Path Setup
- Initialize Empty List: An empty list `files` is created to store the paths of found SAS files.
- Walk Through Directories: `os.walk()`. Generate a tuple for each directory it visits: (root, dirs, files).
- Check Each File: Check if '.sas' is in the filename (case-sensitive). If true, append the full path of the file to the `files` list. `os.path.join(r, file)` creates the full path by combining the directory path and filename
- Print Results: Iter through the files list and print each path.

Find the total number of observations from a SAS data set

```
%let dsn=sashelp.class;
%let dsid    = %sysfunc(open(&dsn,i)) ;
%let nobs    = %sysfunc(attrn(&dsid,nlobs));
%let dsclose = %sysfunc(close(&dsid));
%put &=nobs;
```

How to locate the library path

```
libname work list;
proc options option=work;
run;

%put %sysfunc(getoption(work));
%put %sysfunc(getoption(sasuser));

%put %sysfunc(pathname(work));
%put %sysfunc(pathname(sasuser));
```

Macro

```
dm "log; clear; output; clear; odsresults; clear;";
proc sql noprint;
    select distinct trim(name)
        INTO :AFR_clist separated by ','
        FROM SASHELP.demographics
        where region="AFR";
quit;

%put NOTE: Nonmacro processing region AFR;
%put NOTE: Processing region AFR;
%put Number of countries for AFR_clist = &sqlobs;
%put &=AFR_clist;
%put NOTE: End of region AFR processing;
%put ;

* Macro approach 1;
%macro runit(rname);
proc sql noprint;
    select distinct trim(name)
        INTO :&rname._clist separated by ','
        FROM SASHELP.demographics
        where region="&rname";
quit;
%put NOTE: Processing region &rname using a macro with positional parameter;
%put &rname._clist = %superq(&rname._clist);
%put Number of countries for &rname._clist = &sqlobs;
%put NOTE: End of region &rname processing;
%mend runit;
%runit(AFR)
%runit(AMR)
%runit(EMR)
%runit(SEAR)
%runit(WPR)
%runit(EUR)
```

```
* Macro approach 2;
%macro process_region(rname);
    proc sql noprint;
        select distinct trim(name)
            into :&rname._clist separated by ","
            from SASHELP.demographics
            where region="&rname";
    quit;
    %put NOTE: Processing region &rname using a macro Loop;
    %put NOTE: SQL Observation count = &sqlobs;
    %put &rname._clist = %superq(&rname._clist);
    %put NOTE: End of region &rname processing;
%mend process_region;
%macro run_all;
    %let regions = AFR AMR EMR SEAR WPR EUR;
    %do i = 1 %to %sysfunc(countw(&regions));
        %let region = %scan(&regions, &i);
        %process_region(&region)
    %end;
%mend run_all;
%run_all
```

```
* Macro approach 3;
%macro process_region(rname);
    proc sql noprint;
        select distinct trim(name)
            into :&rname._clist separated by ","
            from SASHELP.demographics
            where region="&rname";
    quit;
    %put NOTE: Processing region &rname using macro with CALL EXECUTE;
       %put NOTE: SQL Observation count = &sqlobs;
    %put &rname._clist = %superq(&rname._clist);
    %put NOTE: End of region &rname processing;
%mend process_region;
data _null_;
    length region $4;
    array regions[6] $4 ('AFR' 'AMR' 'EMR' 'SEAR' 'WPR' 'EUR');
    do i = 1 to dim(regions);
        region = regions[i];
        call execute('%nrstr(%process_region(' || region || '));');
    end;
run;
```

HowToExtractDumpAttributesOfVarsOfInterest.md

Retrieve attributes of variables (common prefix) by listing them as values of the NAME column through DICTIONARY.COLUMN.

```
libname PUFMEPS 'c:\Data\MySDS';
proc sql noprint;
select quote(strip(Name))
  into :varlist separated by ' '
  from dictionary.columns
 where libname= "PUFMEPS" and memname = upcase("h209")
    and name like "AD%";
 %let num_vars_AHS08=&sqlobs;
quit;
```

Dump the attributes of the variables of interest into a spreadsheet.

```
ods excel file = 'c:\Data\Var_Doc_Example.xlsx'
options(row_heights="0,0,0,14,0,0" sheet_interval='PROC'
                  sheet_name="AHS08"
                  embedded_titles="yes");
proc sql;
title "Displaying attributes of %sysfunc(left(&num_vars_AHS08)) variables for
Measure AHS08";
create table AHS08_vars as
select name, type, label
  from dictionary.columns
 where libname= "PUFMEPS" and memname = upcase("h209")
    and name in (&varlist)
              order by name;
  select monotonic() as Number, name, label
     from AHS08_vars;
quit;
ods excel close;
```

Retrieve attributes of variables by listing them as values of the NAME column through DICTIONARY.COLUMN

```
proc sql noprint;
select quote(strip(Name))
  into :varlist separated by ' '
  from dictionary.columns
 where libname= "PUFMEPS" and memname = upcase("h209")
    and name in ('DUPERSID', 'VARSTR', 'VARPSU');
 %let num_vars=&sqlobs;
quit;
```

Retrieve attributes of variables by listing them as values of the NAME column through DICTIONARY.COLUMN where the values are the value of a macro variable.

```
%let mv=  VARSTR VARPSU perwt18f saqwt18f ADFLST42  AGELAST SEX RACETHX
POVCAT18 INSCOV18 ADOFTB42 ADKALC42 ADHOPE42 ADINTR42 ADREST42 ADSAD42
ADMOOD42 ADNERV42 ADPRST42;
%let q_mv=%unquote(%str(%')%qsysfunc(tranwrd(%sysfunc(compbl(&mv)),%str( ),'
'))%str(%'));
%put &q_mv;

proc sql noprint;
select quote(strip(Name))
  into :varlist separated by ' '
  from dictionary.columns
 where libname= "PUFMEPS" and memname = upcase("h209")
    and name in (&q_mv);
%let num_vars = &sqlobs;
quit;
%put &varlist;
%PUT &SQLOBS;
%put _user_;
```

Dump the attributes of the variables of interest into a spreadsheet.

```
ods excel file = 'c:\Data\Selected_SAQ_vars.xlsx'
options(row_heights="0,0,0,14,0,0" sheet_interval='PROC'
                  sheet_name="SAQ"
                  embedded_titles="yes");
proc sql;
title "Displaying attributes of %sysfunc(left(&num_vars)) variables from
MEPS-SAQ, 2018";
create table SAQ_vars as
select name, type, label
  from dictionary.columns
 where libname= "PUFMEPS" and memname = upcase("h209")
    and name in (&varlist)
               order by name;
  select monotonic() as Number, name, label
     from SAQ_vars;
quit;
ods excel close;
```