Using Proc SQL to Run SAS® Procedures on a Group of Datasets within a Single Directory
Diana Ventura, Harvard University, Center for Biostatistics in AIDS Research, Boston, MA

ABSTRACT
Sometimes it is necessary to analyze a group of datasets in the same way. In this case it makes sense to create a macro variable that represents the dataset names. Proc SQL DICTIONARY TABLES contains a variable named memname. The memname variable can be read as a macro variable. The macro variable allows SAS to dynamically generate the dataset names so that SAS procedures can be run without typing the individual names of each dataset. This is a helpful utility that can be used to perform a wide range of SAS procedures with much less typing and repetitive code.

INTRODUCTION
This paper presents a macro that is a helpful utility for running SAS procedures on a group of datasets. Why create a macro variable that represents the names of datasets? Sometimes we have many datasets on which we need to perform the same tasks. Datasets need to be manipulated, frequencies need to be run, graphs need to be generated, and then perhaps all the data needs to exported to Microsoft Excel or some other source. Using memname from the DICTIONARY TABLES dataset as a macro variable allows code to be generated code dynamically eliminating the time and effort necessary to type all the dataset names by hand. Creating the macro variable also creates re-usable code so that a different group of datasets can be analyzed quickly and easily by changing LIBNAME for the group of datasets. Capturing the names of the datasets into a macro variable and using that variable in analysis may seem complicated but it can be accomplish in three easy steps: getting the dataset names into SAS, reading the dataset names into macro variables, and using the macro variables for analysis.

CONCEPTUALLY HOW DOES THIS MACRO WORK

Get dataset names into SAS®
Group of datasets
clindata
demodata
newdata

Read dataset names into macro variables
Macro variables
dsnname1
dsnname2
dsnname3

Use macro variables for Analysis
proc contents
data=&dsnname;i;
run;

THE CODE
Just a few lines of code are needed to be able to run a set of SAS procedures or data step manipulations on a group of datasets.

```sas
%macro group1(lib);
/* The first SQL procedure obtains the count for the second SQL procedure */
proc sql noprint;
   select count(*) into :count
   from dictionary.tables
   where libname=%upcase("&lib");
quit;
```
/
The second SQL procedure obtains the memname variable */
proc sql noprint;
  select memname into :dsname1 - :dsname%TRIM(%LEFT(&count))
    from dictionary.tables
    where libname=%upcase("&lib");
quit;

%do i=1 %to &count;
  proc contents data=&lib..&&dsname&i; run;
%end;
%mend;
%group1(work);

GETTING DATASET NAMES & COUNT VARIABLE
Using Proc SQL to get dataset names requires two separate SQL procedures. The first procedure is used to obtain the total number of datasets in the directory. The second is used to obtain the names of the datasets. The total number of datasets is used in the second SQL procedure as a macro variable and therefore it must be run separately and before the names are obtained.

Step 1: Obtain count variable for the total number of datasets in the directory

proc sql noprint;
  select count(*) into :count
    from dictionary.tables
    where libname=%upcase("&lib");
quit;

<table>
<thead>
<tr>
<th>&amp;lib Macro</th>
<th>Count Macro</th>
<th>Datasets in the directory</th>
</tr>
</thead>
</table>
| WORK       | 3           | clindata
|            |             | demodata
|            |             | newdata

Note: A WHERE statement must be used to specify the directory location of the specific datasets to be included in the analysis. The DICTIONARY TABLES captures runtime information on all tables, indexes, macros used during a SAS session to avoid confusion a where statement is used to capture information concerning a group of datasets in a single directory.

Step 2: Get Dataset names into a macro variable

proc sql noprint;
  select memname into :dsname1 - :dsname%TRIM(%LEFT(&count))
    from dictionary.tables
    where libname=%upcase("&lib");
quit;
According to our above example with three datasets the following statement: `select memname into :dsname1 - :dsname%TRIM(%LEFT(&count))` creates three macro variables one for each dataset in the directory.

<table>
<thead>
<tr>
<th>Dataset Name</th>
<th>Macro Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clindata</td>
<td>dsname1</td>
</tr>
<tr>
<td>Demodata</td>
<td>dsname2</td>
</tr>
<tr>
<td>Newdata</td>
<td>dsname3</td>
</tr>
</tbody>
</table>

**USING THE MACRO FOR ANALYSIS**

```%do i=1 %to &count;
proc contents data=&lib..&&dsname&i; run;
%end;
%mend;
%group1(work);```

An iterative DO LOOP is used to move the macro to perform analysis on each of the datasets in the specified directory. The statement `%do i=1 %to &count;` sets up the loop to move from 1 to the total count determined in the previous PROC SQL step.

<table>
<thead>
<tr>
<th>Code</th>
<th>Resolves to</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>%do i=1 %to &amp;count;</code></td>
<td>Do 1 to 3</td>
</tr>
<tr>
<td><code>proc contents data=&amp;lib..&amp;&amp;dsname&amp;i;</code></td>
<td>Work</td>
</tr>
</tbody>
</table>

So macro simulates the following code:

```proc contents data=work.clindata; run;
proc contents data=work.demodata; run;
proc contents data=work.newdata; run;```

This macro can do more than just Proc CONTENTS. The macro variable created from memname can be used in the same way dataset names are used. This macro code can be used anywhere a dataset name is used provide that the variables will make sense in the code. Care must be used in setting up analysis. For example, if you had a group of datasets to manipulate the datasets must have the variables used in common or an error will occur and “variable not found” message will appear in the log. The macro works great for manipulating groups of datasets in the same way, printing data and generating of export files. SAS procedures and DATASTEPS need to be placed after the `%do I=1 %to &count;` statement.

For example:

```%do I=1 %to &count;
data &lib2..&&dsname&I;
set &lib..&&dsname&I;
more code ... more code ...
run;```
After data manipulations perhaps you would want to compare the datasets. This macro code works well with Proc COMPARE.

```
proc sort data=&lib..&&dsname&i;
  by id;
run;

proc sort data=&lib2..&&dsname&i;
  by id;
run;

proc compare base=&lib..&&dsname&i compare=&lib2..&&dsname&i novalues;
  title 'Comparing Data Sets Original Group with New Group';
run;
```

Once the datasets are compared perhaps they need to be converted into external files. A series of external files can be created in seconds by placing an EXPORT procedure inside the iterative DO LOOP.

```
%do I=1 %to &count;
  proc export
    data=&lib2..&&dsname&i
    outfile="C:\myoutput\&&dsname&i...csv"
    replace;
run;
```

**CONCLUSION**

Using Proc SQL’s DICTIONARY TABLES to create macro variables can save the time it takes to generate code. A few lines of reusable code can save hours of typing. Anywhere a dataset name is used the macro variables can be used in its place. This allows many datasets to be manipulated and analyzed at one time. The macro code presented here eliminates the need for pages of repetitive code and allows groups of dataset to be manipulated with ease.

**CONTACT INFORMATION**

Your comments and questions are encouraged. Contact the author at:

Diana Ventura  
Center for Biostatistics and AIDS Research  
Harvard University School of Public Health  
dventura@sdac.harvard.edu

**TRADEMARK INFORMATION**

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.