PharmaSUG 2015 – Paper QT37

I/O, I/O, It's Off To Work We Go: Digging Into the ATTRC Function

Karleen Beaver, PPD, Morrisville, NC

ABSTRACT

Several file I/O functions allow the SAS® programmer to gain data set properties directly without running a procedure on the data set or retrieving the data set observations. These functions can be called in a DATA step or can be used in open code. This paper illustrates the improvements in efficiency and capability when using the file I/O function ATTRC, specifically the *attr-name* parameter values LABEL and SORTEDBY, compared to running the CONTENTS procedure and accessing DICTIONARY tables in the SQL procedure.

INTRODUCTION

I/O functions provide access to data set properties without the need to compile and execute code or access data set observations.

This paper explores the creation of a macro that takes an input data set and performs manipulations and calculations. When the macro is done executing, the output data set is required to have the same sort attached and the same label as the input data set, if any were assigned to the input dataset at the time the macro was called.

DETERMINING DATA SET LABEL

One way to determine the data set label of the input data set is to use PROC CONTENTS with ODS OUTPUT statement (ods output attributes=attr). Below, Table 1 shows the first 3 columns of the resulting data set.

```
proc contents data = mydata;
   ods output attributes=attr;
run;
```

Member	Label1	cValue1
WORK.MYDATA	Data Set Name	WORK.MYDATA
WORK.MYDATA	Member Type	DATA
WORK.MYDATA	Engine	V9
WORK.MYDATA	Created	Saturday, March 14, 2015 12:30:20 PM
WORK.MYDATA	Last Modified	Saturday, March 14, 2015 12:30:20 PM
WORK.MYDATA	Protection	
WORK.MYDATA	Data Set Type	
WORK.MYDATA	Label	All my data
WORK.MYDATA	Data Representation	WINDOWS_32
WORK.MYDATA	Encoding	wlatin1 Western (Windows)

Table 1. CONTENTS Procedure: Attributes data set from ODS OUTPUT statement

In order to utilize the data set label, the ATTR data set must be accessed and the data set label assigned to a macro variable.

```
data _null_;
   set attr;
   where Label1 = 'Label';
   call symput('label',cValue1);
run;
```

Another way to gather the label information is from the MEMLABEL variable in DICTIONARY.TABLES.

```
proc sql noprint;
   select memlabel into :label
   from dictionary.tables
   where libname='WORK' and
         memname='MYDATA';
quit;
```

This code is simpler than the PROC CONTENTS method since it requires only one step to put the data set label into a macro variable. Though this method's code is simpler in number of steps, it will often be slower than using PROC CONTENTS, particularly when a large number of LIBREFS are assigned.

The last method we will look at to find the data set label is using the LABEL *attr-name* parameter of the ATTRC function.

```
%let dsid=%sysfunc(open(mydata));
%let mylabel=%sysfunc(attrc(&dsid,label));
%let rc=%sysfunc(close(&dsid));
```

If you are unfamiliar with this method, it may look intimidating. The OPEN function opens the data set and assigns the data set identifier to the macro variable DSID. The ATTRC function uses the data set identifier and finds the data set label. The data set must then be closed. No data set observations are accessed and no code is placed on the input stack, so there is no process time. Additionally, the data set label is put into a macro variable in only 3 lines of code.

These I/O functions can be used to create a macro function that returns the data set label. The macro function can then be called in-line.

```
%macro getlabel(dsn=);
   %let dsid=%sysfunc(open(&dsn));
   %sysfunc(attrc(&dsid,label))
   %let rc=%sysfunc(close(&dsid));
%mend;

proc sort data=ds1 out=ds2(label="%getlabel(dsn=mydata)");
   by var1;
run;
```

This macro function could then be called several times on different data sets without creating new macro variables for each data set label.

One caveat to consider is if the input data set did not have a label assigned. In both the PROC CONTENTS and DICTIONARY tables method, if there is no label assigned, the label macro variable will be created with some spaces as the value. The ATTRC method will return a truly blank value. Interestingly, assigning a data set label with (label="") will assign the data set label to ' "" ', whereas assigning a data set label with spaces inside quotation marks will remove a data set label or assign no label. To avoid assigning the label as quotation marks, a check should be added before assigning the label.

```
%if %getlabel(dsn=mydata) ne %then %do;
   proc sort data=ds1 out=ds2(label="%getlabel(dsn=mydata)");
          by age;
   run;
%end;
```

Since the %getlabel macro function uses I/O functions, the added check requires no additional run time.

DETERMINING DATA SET SORT

Similarly to finding the data set label, PROC CONTENTS with ODS OUTPUT statement (ods output sortedby=sortedby) finds the sort assigned to the data set. Below, Table 2 shows the first 3 columns of the resulting data set.

```
proc contents data = mydata;
   ods output sortedby = sortedby;
run;
```

Member	Label1	cValue1
WORK.MYDATA	Sortedby	study age
WORK.MYDATA	Validated	YES
WORK.MYDATA	Character Set	ANSI

Table 2. CONTENTS Procedure: SORTEDBY data set created by ODS OUTPUT statement

Again, in order to utilize the data set sort, the SORTEDBY data set must be accessed and the data set sort assigned to a macro variable.

```
data _null_;
   set sortedby;
   where Label1 = 'Sortedby';
   call symput('mysort',cValue1);
run;
```

The SORTEDBY variable in DICTIONARY.COLUMNS also provides the desired information. Below, Table 3 shows the columns of interest of the resulting data set.

```
proc sql noprint;
   create table mysort as
   select * from dictionary.columns
   where libname='WORK' and
         memname='MYDATA';
quit;
```

name	varnum	sortedby
study	1	1
subj	2	0
age	3	2
sex	4	0
result	5	0

Table 3. DICTIONARY tables: SORTEDBY variable from DICTIONARY tables

The information is spread across several rows, but only one step is needed to get the data set sort into a macro variable.

```
proc sql noprint;
   select name
   into :mysort separated by ' '
   from dictionary.columns
   where libname='WORK'
         and memname='MYDATA'
         and sortedby > 0
   order by sortedby;
quit;
```

Similarly to finding the label, this code is simpler than the PROC CONTENTS method since it requires only one step to put the data set sort into a macro variable. Again, though this method requires simpler code, it will often require more process time because of the DICTIONARY scan.

Once again, the SORTEDBY *attr-name* parameter of the ATTRC function also allows access to the data set sort assigned.

```
%let dsid=%sysfunc(open(mydata));
%let mysort=%sysfunc(attrc(&dsid,sortedby));
%let rc=%sysfunc(close(&dsid));
```

As with the data set label, a macro function that returns the data set sort can be created and called in-line.

```
%macro getsort(dsn=);
   %let dsid=%sysfunc(open(&dsn));
   %sysfunc(attrc(&dsid,sortedby))
   %let rc=%sysfunc(close(&dsid));
%mend;

proc sort data=ds1 out=ds2;
   by %getsort(dsn=mydata);
run;
```

There remains no process time and the size of the data set does not have to be considered with this method.

NO SORT ASSIGNED TO DATA SET

A macro that takes in an input data set from a user needs to account for multiple situations, including an input data set that does not have a label or sort assigned. In the above methods for determining data set label, a data set with no label would not lead to actual errors. However, a data set with no sort assigned would lead to errors with the PROC CONTENTS method of determining data set sort.

```
proc contents data = mydata;
   ods output sortedby = sortedby;
run;
```

With this code, if there is no sort assigned to the data set, a log warning message appears:

```
WARNING: Output 'sortedby' was not created.  Make sure that the output object name, label, or
         path is spelled correctly.  Also, verify that the appropriate procedure options are used
         to produce the requested output object.  For example, verify that the NOPRINT option is
         not used.
```

```
data _null_;
   set sortedby;
   where Label1 = 'Sortedby';
   call symput('mysort',cValue1);
run;
```

If the SORTEDBY data set is not created, the above code errors out and the macro variable, MYSORT, will not be created. Consequently, trying to reference the MYSORT macro variable in a PROC SORT leads to further warnings and errors.

The DICTIONARY tables method would have similar errors if the data set sort was not assigned.

```
proc sql noprint;
   select name
   into :mysort separated by ' '
   from dictionary.columns
   where libname='WORK'
         and memname='MYDATA'
         and sortedby > 0
   order by sortedby;
quit;
```

If there is no sort assigned to the data set, MYDATA, then the macro variable, MYSORT, will not be created, and the same issues will be seen as with the PROC CONTENTS method.

When using the SORTEDBY *attr-name* parameter of the ATTRC function, the macro variable will still be created even if no sort is assigned to the data set so a simple error trap solves the issue.

```
%let dsid=%sysfunc(open(mydata));
%let mysort=%sysfunc(attrc(&dsid,sortedby));
%let rc=%sysfunc(close(&dsid));

%if &mysort ne %then %do;
   proc sort data=ds1 out=ds2;
          by &mysort;
   run;
%end;
```

In this case, the macro variable MYSORT is blank, so the PROC SORT code will not be executed as the %do loop is not entered.

A simple error trap also works when using a macro function.

```
%macro getsort(dsn=);
```

```
    %let dsid=%sysfunc(open(&dsn));
    %sysfunc(attrc(&dsid,sortedby))
    %let rc=%sysfunc(close(&dsid));
%mend;

%if %getsort(dsn=mydata) ne %then %do;
proc sort data=ds1 out=ds2;
    by %getsort(dsn=mydata);
run;
%end;
```

Similarly to checking for a data set label, this check for assigned sort does not add process time.

If no other action is desired if the data set sort is blank, this method suffices. However, if another action is desired, the IFC function could add more functionality to the macro.

```
%macro getsort(dsn=);
    %let dsid=%sysfunc(open(&dsn));
     ** if sort is not missing, return the sort variables, otherwise return _all_ ;
    %sysfunc(ifc(%sysfunc(attrc(&dsid,sortedby)) ne ,
                 %sysfunc(attrc(&dsid,sortedby)),
                 _all_))
    %let rc=%sysfunc(close(&dsid));
%mend;
```

PROCESSING DATA SET LABEL AND SORT ON ALL DATA SETS IN A LIBRARY

Another scenario to consider is creating a macro that takes in a library name and processes all data sets within the library while maintaining the original data set labels and sorts.

METHOD 1: PROC CONTENTS

Using PROC CONTENTS, the data set label and sort on all data sets in the library are obtained in one step. Here, all data sets in the WORK library are processed, but the library name could be passed as a macro parameter.

```
proc contents data = work._all_;
    ods output sortedby = sortedby attributes=attr;
run;
```

The labels and sort variables, as well as the data set names, are put into macro variables that can be looped through in a later step.

```
data _null_;
    merge attr(where=(Label1='Label') rename=(cValue1=dslabel))
    sortedby(where=(Label1='Sortedby') rename=(cValue1=sortvars)) end=eof;
    by member;
    call symput(compress('ds'||put(_n_,3.)),substr(strip(member),6));
    call symput(compress('label'||put(_n_,3.)),strip(dslabel));
    call symput(compress('sort'||put(_n_,3.)),strip(sortvars));
    if eof then call symputx('dsnum',_n_);
run;
```

Now all data sets in the library are processed in a %do loop, assigning the original data set label and sort.

```
%do i = 1 %to &dsnum;
    %if &&sort&i ne and &&label&i ne %then %do;
           proc sort data=&&ds&i out=outlib.&&ds&i(label="&&label&i");
                  by &&sort&i;
           run;
    %end;
%end;
```

METHOD 2: DICTIONARY tables and I/O functions

A list of the names of all the data sets in the library are set into a macro variable using DICTIONARY.MEMBERS.

```
proc sql noprint;
    select memname
    into :mems separated by ','
    from dictionary.members
    where libname='WORK'
          and memtype='DATA';
quit;

%let dsnum = &sqlobs; ** save this for later use;
```

The previously defined macro functions provide the sort variables and label assignment.

```
%do i = 1 %to &dsnum;
    %let ds = %qscan(%bquote(&mems),&i,%str(,)); ** &mems is a comma delimited list
                        of data set names, need to get one data set name at a time ;
    %if %getlabel(dsn=&ds) ne and %getsort(dsn=&ds) ne %then %do;
           proc sort data=&ds out=outlib.&ds(label="%getlabel(dsn=&ds)");
                  by %getsort(dsn=&ds);
           run;
    %end;
%end;
```

The macros that loop through all data sets in a library in method 1 and method 2 have similar run times, however, retrieving the list of data sets from DICTIONARY.MEMBERS has shorter run time than running PROC CONTENTS. Additionally, method 1 requires an extra DATA step to put the needed information into macro variables.

CONCLUSION

The ATTRC file I/O function presents the programmer with an alternative method to finding data set label and sort and is often times more efficient than other methods. Use of the ATTRC function can improve run time and code efficiencies, especially when determining data set sort. When no data set sort is assigned to the input data set, the ATTRC function allows the user to write robust code with very little error trapping.

REFERENCES

Simon, J., et. al. 2013. *SAS Macro Language 2: Advanced Techniques*. 4-30 – 4-34. SAS Institute Inc. Cary, NC, USA.

ACKNOWLEDGEMENTS

I would like to thank my colleague, Ken Borowiak, and my husband for reviewing my paper and providing very helpful feedback.

CONTACT INFORMATION

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

Karleen Beaver
PPD
3900 N Paramount Parkway
Morrisville, NC 27560
Karleen.Beaver@ppdi.com

DISCLAIMER

The contents of this paper are the work of the author and do not necessarily represent the opinions, recommendations, or practices of PPD.