Paper CT-01

SAS® Macros and the SAS® DATASETS Procedure –

An automated approach to data set management and manipulation

Chris Alexander, RTI International, Research Triangle Park, NC

ABSTRACT

The combination of Macro functionality and the SAS® DATASETS procedure provides a simple, straightforward approach to automating data set manipulation and management. For users who wish to perform common tasks across groups of data sets, the DATASETS procedure is the tool to programmatically identify which data set(s) to work with. When it is combined with the Macro facility, users can write compact, maintenance-friendly (or even maintenance-free) code to manipulate the data set(s) targeted by the DATASETS procedure in any number of ways. This paper assumes a basic knowledge of the DATA step and SAS® Macro language.

INTRODUCTION

SAS users often work with collections of data sets that are stored in a common location, be it on a hard drive, server, or otherwise. The DATASETS procedure can be utilized to identify what data sets exist in a given location, thereby relieving the programmer of the need to explicitly address each data set individually in code. Most alternative approaches require the programmer to keep track of which data sets are new to a given collection over time and which have been removed because they are no longer needed. The burden of adjusting the code accordingly then falls on the user. If very different actions need to be taken on various data sets in a collection, then manually catering code to each data set is likely a necessity, in which case the approach described in this paper would not be recommended. In instances where the same action or actions need to be taken on the data sets in a collection, however, users stand to gain quite a bit by calling on the DATASETS procedure to help automate their processes.

- **Vocabulary note:** The word “collection(s)” is used in this paper to refer to a grouping or groupings of data sets that a given user is targeting.

PROC DATASETS OVERVIEW

The DATASETS procedure offers users a wide array of functionality that can be applied to all types of SAS files. This paper will focus mainly on the **CONTENTS** statement which only caters to data sets.

- The **CONTENTS** statement, as described in the SAS procedure guide, “describes the contents of one or more SAS data sets and prints the directory of the SAS library”. When used together with the **OUT=** option, the contents of a given library can be captured in a SAS data set in addition to or instead of the printed contents.
- The **MEMTYPE=** option restricts the focus of the DATASETS procedure call to a certain type of SAS file. Since the scope of this paper is limited to data sets, MEMTYPE=DATA is specified in the example code.

The following example demonstrates the use of the CONTENTS statement in conjunction with the OUT= option. The DATASETS procedure call ultimately populates a data set named mylib_contents (in the “work directory”) with a de-duplicated list of the data sets that exist in library named myLib.

```
LIBNAME myLib ".";

/* MLB players */
DATA myLib.ds_mlb;
 infile datalines delimiter=',';
 input LastName $ Position $ JerseyNum;
datalines;
 Heyward,Outfield,22
 McCann,Catcher,16
 Smoltz,Pitcher,29
 Glavine,Pitcher,47;
```

```
/* NBA players */
DATA myLib.ds_nba;
 infile datalines delimiter=',';
 input LastName $ Position $ JerseyNum ShoeSize;
datalines;
 Nowitzki,Forward,41,14
 Jordan,Guard,23,13
 Garnett,Center,5,15;


/* NFL players */
DATA myLib.ds_nfl;
 infile datalines delimiter=',';
 input LastName $ Position $ JerseyNum;
datalines;
 Williams,DB,90
 Holt,WR,81
 Rivers,QB,17;

/* Populate mylib_contents with the names of the data sets in myLib */
proc DATASETS memtype=data lib=myLib;
   CONTENTS data=_ALL_ OUT=work.mylib_contents(KEEP=MEMNAME NOBS CRDATE) NOPRINT;
run;

/* The CONTENTS statement outputs a row for each column in each data set, */
/* therefore the SORT procedure is called with the NODUPKEY option set    */
/* in order to obtain a de-duplicated list of the data sets.              */
proc SORT data=work.mylib_contents NODUPKEY;
 by MEMNAME;
run;
```

The CONTENTS statement associated with the DATASETS procedure call outputs a row for every column in each data set. A de-duplicated list of the data sets belonging to the myLib library is needed for this example, therefore, the SORT procedure is called upon with the NODUPKEY option set.

MEMNAME	NOBS	CRDATE
DS_MLB	4	11AUG12:18:56:32
DS_NBA	3	11AUG12:18:56:32
DS_NFL	3	11AUG12:18:56:32

Table 1. mylib_contents data set

The de-duplicated list of data sets in the myLib library can now be referenced via a DATA step which affords users the ability to execute any code of their choosing on each data set programmatically as opposed to manually tailoring separate portions of code to each data set in the collection. The following section provides an example that illustrates this point.

COMBINING PROC DATASETS WITH MACRO CALLS

Building on the example code in the DATASETS overview section above, if a user needed to run the FREQ and SORT procedure on every data set in the myLib library *and* any future data sets that accumulate in that library, that objective can be accomplished in any number of ways. The following illustrates three different approaches, the last of which is in line with this core concept of this paper.

Approach 1: Tailoring individual procedure calls to each data set

Elements of this approach that are not ideal:

- Duplication of code: A simple macro could be utilized to eliminate the duplicated code.
- Maintenance burden: In this example, the user expects the number of data sets in the target library to increase over time. For each data set that gets added to the library, the user will have to manually add code for the associated SORT and FREQ procedure calls.

```
/* MLB players */
proc SORT data=myLib.ds_mlb;
 by JerseyNum;
run;

proc FREQ data=myLib.ds_mlb;
run;

/* NBA players */
proc SORT data=myLib.ds_nba;
 by JerseyNum;
run;

proc FREQ data=myLib.ds_nba;
run;

/* NFL players */
proc SORT data=myLib.ds_nfl;
 by JerseyNum;
run;

proc FREQ data=myLib.ds_nfl;
run;
```

Approach 2: Utilizing the Macro facility

A macro-based approach eliminates the need to write duplicate code for each of the targeted data sets, however, it does not resolve the maintenance burden issue. The user will still have to manually add code in accordance with the addition of data sets to the myLib library over time.

```
%MACRO RunSortAndFreq(DSname=);

   /* Players */
   proc SORT data=myLib.&DSname;
    by JerseyNum;
   run;

   proc FREQ data=myLib.&DSname;
   run;

%MEND;

%RunSortAndFreq(DSname=ds_mlb)
%RunSortAndFreq(DSname=ds_nba)
%RunSortAndFreq(DSname=ds_nfl)
```

For example, if a data set named “ds_nhl” is added to the myLib library and that the user wishes to execute the RunSortAndFreq macro for it, as is being done for the other data sets in the collection, the user will have to manually add the line of code below in their program. In instances where data sets are being added to and/or subtracted from the targeted library quite often, this sort of manual program maintenance can become quite burdensome.

```
%RunSortAndFreq(DSname=ds_nhl)
```

Approach 3: Utilizing the Macro facility in concert with proc DATASETS

Utilizing the DATASETS procedure allows the user to retrieve a list of the data sets in the myLib library. This approach does require additional lines of code, however, it relieves the user of the maintenance burden referenced in the first two approaches described above. No code adjustments are necessary to accommodate the addition and/or subtraction of data sets in the target library over time.

```
%MACRO RunSortAndFreq(DSname=);

   /* Players */
```

```
    proc SORT data=myLib.&DSname;
     by JerseyNum;
    run;

    proc FREQ data=myLib.&DSname;
    run;

%MEND;

/* Populate mylib_contents with the names of the data sets in myLib */
proc DATASETS memtype=data lib=myLib;
   CONTENTS data=_ALL_ OUT=work.mylib_contents(KEEP=MEMNAME NOBS CRDATE) NOPRINT;
run;

proc sort data=work.mylib_contents NODUPKEY;
 by MEMNAME;
run;

/* Execute the RunSortAndFreq macro on each data set in myLib */
DATA _null_;
set work.mylib_contents;
LENGTH MacroCall $300.;
MacroCall = '%RunSortAndFreq(DSname=' || trim(left(MEMNAME)) || ');';
  CALL EXECUTE(MacroCall);
run;
```

The CALL EXECUTE line in the example DATA step above effectively executes the following code in an automated fashion:

```
%RunSortAndFreq(DSname=ds_mlb)
%RunSortAndFreq(DSname=ds_nba)
%RunSortAndFreq(DSname=ds_nfl)
```

Approach 3, expanded: Narrowing focus on target data sets

For many users, the need to execute the same code on every single data set in a given library will not prove to be practical. More than likely, users will instead wish to focus on a subset of data sets within the target library. Adjusting the code to tease out the desired subset of data sets is often simply a matter of adding if/then statements to the DATA step that executes the macro call(s). Several examples are shown in the code below.

- Example 1: subsets the collection of data sets to those whose names include the string “ds_m”.
- Example 2: subsets the collection of data sets to those that are more than 5 days old.
- Example 3: subsets the collection of data sets to those that have more than 3 observations.

```
/* Execute the RunSortAndFreq macro on a subset of the data sets in myLib */
DATA _null_;
set work.mylib_contents;
LENGTH MacroCall $300.;

If INDEX(MEMNAME,'ds_m') > 0 Then Do; /* Example 1 */
If CRDATE > (DATE() - 5) Then Do;     /* Example 2 */
If NOBS >= 4 Then Do;                 /* Example 3 */

    MacroCall = '%RunSortAndFreq(DSname=' || trim(left(MEMNAME)) || ');';
      CALL EXECUTE(MacroCall);
    run;
End;
```

CONCLUSION

When working with very small and/or stable collections of data sets, utilizing the approach outlined in this paper may not prove to be ideal. When working with relatively large and/or dynamic collections of data sets, however, utilizing PROC DATASETS in concert with the SAS macro facility to automatically identify a given collection of data sets can

significantly reduce the amount of code needed to manipulate that collection as well as significantly reducing the level of effort necessary to maintain the facilitating program.

REFERENCES

- Base SAS® Procedures Guide

CONTACT INFORMATION

Name: Chris Alexander
Enterprise: RTI International
Address: 3040 East Cornwallis Road
City, State ZIP: Research Triangle Park, NC 27709
Work Phone: 919-316-3816
E-mail: cpa@rti.org