Paper BB-09-2013

OPTIMIZE YOUR DELETE

Brad Richardson, SAS Institute Inc., Cary, NC

ABSTRACT

Have you deleted a data set or two from a library that contains thousands of members using PROC DATASETS? If so, you probably have witnessed some wait time. To maximize performance, we have reinstated PROC DELETE as a SAS-supported procedure. One of the main differences between PROC DELETE deletion methods versus PROC DATASETS DELETE is that PROC DELETE does not need an in-memory directory to delete a dataset. So what does this mean exactly? This paper will explain all.

INTRODUCTION

The purpose of this paper is not to teach you the basics of PROC DELETE and PROC DATASETS, but instead to show the new features available for PROC DELETE and discuss the optimization of the deletion processes. We will examine when PROC DELETE should be used over PROC DATASETS and vice versa. The revival of PROC DELETE in SAS 9.4 has provided several new features. Also, by studying the PROC DELETE process, the developers were able to improve other Base Utility Procedures.

PROC DATASETS

PROC DATASETS is an interactive procedure which allows you to manage members in your SAS library. Since DATASETS is an interactive procedure it is able to contain the functionality of several stand-alone procedures and some functions that are exclusive to the procedure. In most cases, having all of this functionality under one umbrella decreases the overhead when an entire library is altered. The procedure was created to handle libraries, not just a few members within the library or a single deletion. At the time when PROC DATASETS was created memory was not as easily available and as inexpensive as it is today. Many libraries did not contain a massive amount of data sets. Therefore, to cut down on processing time, the DATASETS procedure has a directory, or list of members in the library which it keeps in memory this is called the in-memory directory. The benefit of keeping a directory in memory is that, when a library needs to be modified, the DATASETS procedure only has to access the disk to perform the requested actions and the presence of the members need never be queried.

WHAT IS THE IN-MEMORY DIRECTORY?

The in-memory directory is a list which contains SAS library members and their auxiliary files, such as indexes, audit trails, and extended attributes. The directory is created so that PROC DATASETS only has to hit the disk once to determine the existence of the library members. The directory is created by reading the names of all the files in the directory associated with the libname used and determining which members of SAS library. PROC DATASETS stores this list in memory and uses it to act on for any of the functions it provides, such as deleting, renaming, aging, appending, copying, etc. PROC DATASETS creates an in-memory directory each time when it initializes. When PROC DATASETS was originally written, the overhead of creating an in-memory directory required far less time than accessing the hard disk multiple times. Over time this has changed due to the rapid improvement of condensing physical memory and increasing memory capacity.

Partially due to the increase in easily obtainable memory capacity, there is an emergent requirement to store and retain a more data. Therefore, the number of members stored in SAS libraries has increased dramatically over the years. On the other hand the bus to transfer this massive data has not grown as quickly; as a result there are longer wait times to create the in-memory directories. If you have libraries that have thousands of members, it may not be optimal for DATASETS to create an in-memory directory. The overhead of creating an in-memory directory to manipulate many members in a library is still an optimal algorithm, but if you are going to modify only a few members within a library then you may want to consider a different approach. So what should you do when you only need a few members modified?

WELCOME BACK PROC DELETE!

To efficiently modify a few members within a library, utilize some of the singular function procedures. One of the main

complaints we heard from customers was removing datasets. To resolve the problem; PROC DELETE has been reinstated as a supported procedure. Since PROC DELETE's sole purpose is to permanently eliminate library members, there is no need to build an in-memory directory. PROC DELETE is meant to delete members within a library, but not an entire library. So, unlike PROC DATASETS, PROC DELETE does not check to see if the selected data set actually exists before trying to remove the file. Only after the attempt to erase the file will PROC DELETE know whether the dataset actually existed, since there is no overhead to get the member names from disk, transport the data over the bus, and create an in-memory directory, it speeds the deletion process up. The unsupported DELETE procedure lacked a lot of functionality that PROC DATASETS supported, so to bring it up-to-date some new features were added to DELETE.

WHAT ARE THE NEW FEATURES?

Considering that PROC DELETE has not officially been supported by SAS since the late 90s, there were a number of new features that needed to be added to bring the procedure up-to-date. The goal of the new features was to have PROC DELETE mirror the behaviors of PROC DATASETS' DELETE statement as much as possible. Some of the new features added to PROC DELETE include:

- Numeric suffix lists
- LIB=libref option
- MEMTYPE=
- ENCRYPTKEY=(encryptkey)

One of the new functionalities added to PROC DELETE is the numeric suffix list. The numeric suffix list feature in PROC DELETE works the same as that in the PROC DATASETS DELETE statement. The list allows the user to specify a range of datasets to be removed without having to explicitly list all of the datasets. The syntax requires that you have a series of datasets with the same name, ending in a numeric value.

```
proc delete data = A1-A3;
run;
quit;
```

From the example above datasets A1, A2 and A3 in the WORK library will be deleted if they exist, but if they do not exist then an error message will be generated saying that the file does not exist. Also, if the first dataset numeric suffix contains a leading zero, then the last dataset name must also have a leading zero.

Now we will look at the LIB= option. The LIB= option identifies the library containing the data sets. Previously PROC DELETE would require a two level name for each dataset if it was not in the WORK library, but now the LIB= option allows for a one level name on any library. Only one library name can be specified for this option. The LIB= option can be used in conjunction with any other feature. The two level name is still accepted in PROC DELETE.

```
libname mylib ".";
proc delete lib=mylib data = A1-A3;
run;
```

Next is the MEMTYPE= option which provides flexibility in what member types are being removed. In the past PROC DELETE has not allowed you to specify what memtype you would like to delete. Since the default MEMTYPE was DATA back when PROC DELETE was previously supported, the procedure would only delete members with the DATA memtype. Currently the default MEMTYPE is still DATA, but you can override the member type with any of the SAS supported memtypes.

The ENCRYPTKEY= option allows the SAS data set to be encrypted using AES encryption. There are two different kinds of encryption available in SAS 9.4 SAS Proprietary encryption and AES Encryption.

The first kind is SAS Proprietary encryption is automatically provided with Base SAS software. SAS Proprietary encryption uses 32-bit fixed encoding and is appropriate for preventing accidental exposure of information.

```
data abc.foo(encrypt=yes read=rr);

x=1;run;

proc datasets lib=abc nolist;

delete foo;
```

```
run;
quit;
```

The second kind of encryption is AES, provided with SAS/SECURE software. AES provides a stronger encryption that can be up to 64 characters long; it requires that the ENCRYPTKEY be specified each time the data set is opened.

```
data lib1.x1(encrypt=aes encryptkey=key);
x=1;
run;

proc delete lib=lib1 data=x1(encryptkey=key);
run;
quit;
```

PROC DELETE VS PROC DATASETS

In previous sections we discussed the deletion process used by PROC DATASETS and PROC DELETE, as well as why PROC DELETE may require less processing time when removing various datasets. Now let's do a time comparison of the two procedures. In the example below, we created a library containing 10,000 members with the DATA member type. In the first example both PROC DELETE and PROC DATASETS remove one dataset from the library.

```
18   proc datasets lib=mylib;
19                 delete a5674;
20                 run;

NOTE: Deleting MYLIB.A5674 (memtype=DATA).
20 !                   quit;

NOTE: PROCEDURE DATASETS used (Total process time):
      real time           8.63 seconds
      cpu time            8.56 seconds
```

Output 1. Results from the deletion in a large library with PROC DATASETS

```
21   proc delete
22   data=mylib.a5675;
23    run;

NOTE: Deleting MYLIB.A5675 (memtype=DATA).
NOTE: PROCEDURE DELETE used (Total process time):
      real time           0.02 seconds
      cpu time            0.01 seconds
```

```
23 !       quit;
```

Output 2. Results from the deletion in a large library with PROC DELETE

Next, both procedures will delete five datasets from the same library. The library is recreated before the PROC DATASETS test to keep the results in line.

```
2     proc datasets lib=mylib;
NOTE: Writing HTML Body file: sashtml.htm
3              delete a9042;
4              delete a4685;
5              delete a1895;
6              delete a361;
7        run;

NOTE: Deleting MYLIB.A9042 (memtype=DATA).
NOTE: Deleting MYLIB.A4685 (memtype=DATA).
NOTE: Deleting MYLIB.A1895 (memtype=DATA).
NOTE: Deleting MYLIB.A361 (memtype=DATA).
8     quit;

NOTE: PROCEDURE DATASETS used (Total process time):
      real time           26.03 seconds
      cpu time            13.33 seconds
```

Output 3. Results from the deletion of five datasets in a large library with PROC DATASETS

```
2        proc delete
3       data=
4       mylib.a9042
5       mylib.a4685
6       mylib.a1895
7       mylib.a361;run;

NOTE: Deleting MYLIB.A9042 (memtype=DATA).
NOTE: Deleting MYLIB.A4685 (memtype=DATA).
NOTE: Deleting MYLIB.A1895 (memtype=DATA).
NOTE: Deleting MYLIB.A361 (memtype=DATA).
NOTE: PROCEDURE DELETE used (Total process time):
      real time           0.08 seconds
      cpu time            0.01 seconds
```

Output 4. Results from the deletion of five datasets in a large library with PROC DELETE

The first test case was to delete a single dataset from a massive library. Comparing the results of the first test case PROC DATASETS completed the task in 8.63 seconds in real time and 8.56 seconds for cpu time. However PROC

DELETE produces the same results in 0.02 seconds real time and 0.01 seconds in cpu time. Next let's compare, the second test case total processing times; PROC DATASETS completed the task in 26.03 seconds in real time and 13.33 seconds for cpu time. In comparison, DELETE was able to finish the job in 0.08 seconds real time and 0.01 seconds in cpu time. PROC DELETE was able to complete the same tasks, but in a fraction of the time. In both test cases the cpu time decreased by 99.7%~99.9% and the real time was reduce to 99.7%~99.8%. The additional processing time required by PROC DATASETS was used to create the in-memory directory. The library that was created only has a few thousand datasets, which may be comparatively small to the numbers that are being used within large companies. Therefore, using PROC DATASETS to delete a couple of members within a larger library would increase the total processing time, whereas PROC DELETE will keep a consistent processing time.

So you may be asking yourself, why not get rid of the in-memory directory completely? The reason is because there are many benefits to retaining an in-memory directory. Even though PROC DELETE processing time for deletion is faster than PROC DATASETS, there are also negatives to using PROC DELETE.

PROS AND CONS OF PROC DELETE

The return of PROC DELETE has significantly improved the deletion process in terms of time, but in contrast we lose some of the capabilities of PROC DATASETS. If you recall, PROC DATASETS is an interactive procedure and PROC DELETE is a non-interactive procedure. PROC DELETE's only objective is to permanently remove members from the library, whereas PROC DATASETS is a multiple purpose procedure; this is why PROC DATASETS keeps an in-memory directory.

In essence the in-memory directory makes it possible for PROC DATASETS to have much of it functionality. The directory that PROC DATASETS creates and stores in memory allows you to run a series of commands with one procedure. As an illustration, you can rename a member, repair damaged member, and remove a file all in one step. The code is optimized because one procedure is being used. Also the wildcard "*" and all "/" commands cannot be efficiently implemented without the in-memory directory. For PROC DELETE to support this functionality it will have to access the disk multiple times to see if any of the library members meet the criteria, which would be very costly.

Of course, there is a down side to using an in-memory directory as well. Aside from the time required to create the in-memory directory, also the in-memory directory is not fail safe with multiple people accessing the library. Consider an instance when you have two SAS sessions accessing the same library at the same time. One session deletes a dataset called "myExample" their in-memory directory is updated reflecting the change. Now the other session would like to rename "myExample" to "theEx". PROC DATASETS will check the in-memory directory to see if the file exists. Since the user's DATASETS does not access the disk, it shows that the file "myExample" still exists. When RENAME is called an error is surfaced.

PROC COPY OPTIMIZED

As mentioned in previous sections PROC DATASETS keeps an in-memory directory to help process some of the commands, PROC DATASETS is not the only procedure that keeps a directory in memory. PROC COPY also creates and stores a directory in memory. PROC COPY was created to use the in-memory directory for the same reasons PROC DATASETS did. At the time, libraries were not nearly so large as they are today. PROC COPY is often used to copy an entire library. Of course when this occurs, the disk must be accessed to determine all the members that the library contains.

Similar to PROC DELETE, if the desired action is to modify a few members, and then there is no need to create an in-memory directory. In SAS 9.4, the PROC COPY SELECT statement has been optimized to avoid the in-memory directory when possible. The PROC COPY SELECT statement without a MEMTYPE= option will copy any member in the library with the name specified. Therefore, the optimization requires the PROC COPY caller to specify exactly what member types are required. PROC COPY without the optimization knows what member types exist by looking them up in the in-memory directory. The new algorithm is very convenient for those current users that use PROC COPY calls with SELECT and MEMTYPE specified are automatically optimized; without you changing any SAS code. To receive the performance gain, it must be a SELECT copy with a member type specified.

For example, let's take the library that was used in the previous example and copy a couple of datasets to another library. First, we will run the test case using no MEMTYPE= option afterwards the same action will use MEMTYPE=.

```
38    proc copy in=mylib out=templib;
39    select a2542 a9236;run;


NOTE: Copying MYLIB.A2542 to TEMPLIB.A2542 (memtype=DATA).
```

```
NOTE: There were 1 observations read from the data set MYLIB.A2542.
NOTE: The data set TEMPLIB.A2542 has 1 observations and 1 variables.
NOTE: Copying MYLIB.A9236 to TEMPLIB.A9236 (memtype=DATA).
NOTE: There were 1 observations read from the data set MYLIB.A9236.
NOTE: The data set TEMPLIB.A9236 has 1 observations and 1 variables.
NOTE: PROCEDURE COPY used (Total process time):
      real time           0.90 seconds
      cpu time            0.87 seconds
```

Output 5. Results from the copying of two data sets in a large library with SELECT but without MEMTYPE=

```
43   proc copy in=mylib out=templib memtype=data;
44   select a2542 a9236;run;

NOTE: Copying MYLIB.A2542 to TEMPLIB.A2542 (memtype=DATA).
NOTE: There were 1 observations read from the data set MYLIB.A2542.
NOTE: The data set TEMPLIB.A2542 has 1 observations and 1 variables.
NOTE: Copying MYLIB.A9236 to TEMPLIB.A9236 (memtype=DATA).
NOTE: There were 1 observations read from the data set MYLIB.A9236.
NOTE: The data set TEMPLIB.A9236 has 1 observations and 1 variables.
NOTE: PROCEDURE COPY used (Total process time):
      real time           0.03 seconds
      cpu time            0.03 seconds
```

Output 6. Results from the copying of two data sets in a large library with SELECT and MEMTYPE=

The changes to PROC COPY speeds up the operations both for real time and cpu time by 96.6%~96.7%. Similar to PROC DELETE, PROC COPY processing time will not increase when the library size increases.

CONCLUSION

In summary, PROC DELETE's new changes have made it more efficient to remove data sets from the library in SAS 9.4 release. There are a few things to consider when determining whether PROC DELETE is the right tool to delete the data sets. If you are looking to just remove a few data sets from the library, then PROC DELETE is the best option. On the other hand if you need to delete an entire library, then PROC DATASETS is the best tool. By knowing what the differences are between PROC DELETE and PROC DATASETS DELETE deletion process, you can receive a massive performance gain in your SAS application.

ACKNOWLEDGMENTS

Thanks to Diane Olson, Lisa Brown, and Peggy Cavalieri for reviewing this paper. Thanks to Sue Holmes for the ENCRYPTKEY documentation. Thanks to Miguel Bamberger for the SAS code and all of his advice.

CONTACT INFORMATION

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

Brad Richardson
SAS Institute Inc.
500 SAS Campus Drive
Cary NC 27513
Work Phone: (919)677-8000

E-mail: Brad.Richardson@sas.com