Paper 23-27

Programming Tricks For Reducing Storage And Work Space

Curtis A. Smith, Defense Contract Audit Agency, La Mirada, CA.

ABSTRACT

Have you ever had trouble getting a SAS® job to complete, although your SAS code is bug-free? Often, especially when processing large amounts of data, you can exhaust your storage space or the WORK library. Likely, there is a SAS code solution. Herein, the author will share some of his favorite tricks to squeeze more out of storage and WORK library space using SAS code to resolve resource limitations. All of the tricks will use SAS code from base SAS. This paper will cover subjects such as reducing the number of SAS data sets needed in the work space, reducing the size of SAS data sets, and cleaning up the work library. The author will concentrate on the DATASETS, DELETE, SUMMARY, and SORT procedures, the DROP, KEEP, COMPRESS, and OUT options of the Data Step, and the WHERE statement. SAS programmers at any level can benefit from these tips.

INTRODUCTION

In my endeavors to get my SAS jobs to complete when processing large amounts of data, I encounter temporary work space shortages as well as permanent storage space shortages. SAS, of course, uses its WORK library as the place to create and store temporary work data sets. A SAS job can require many temporary SAS data sets, even some that you may not reference in a DATA step or procedure. Of course, you decide where to store permanent SAS data sets. But sometimes, you may not have enough free space for the temporary or permanent SAS data sets you want to create. Exhausting temporary and permanent storage space is common. Many approaches to solve these problems are available. Many solutions involve the operating system. But, I will present solutions that involve only using SAS code and fall into three categories:

- Reduce the size of the SAS data sets
- Reduce the number of SAS data sets
- Clean up the storage space

REDUCE THE SIZE OF THE SAS DATA SETS

There are many ways to reduce the size of SAS data sets that you place in your temporary work space and in permanent storage. Below I share my favorite strategies.

COMPRESSING FILES

SAS can use compression algorithms to compress SAS data sets. This is a process of reducing the amount of space needed to store a SAS data set - it does not affect the data stored within that SAS data set. Using the COMPRESS= system or data set option, any SAS data set created on disk will be compressed. SAS data set compression can greatly reduce the size of SAS data sets. To use the COMPRESS= system or data set option, set the option to either "YES" or "BINARY." (In newer versions of SAS, “CHAR” can be used as an alternative to “YES” with the same result.) The COMPRESS=YES value uses an algorithm that works better with SAS data sets that are primarily comprised of character variables. On the other hand, COMPRESS=BINARY uses a different algorithm that works better with SAS data sets that are primarily comprised of many variables including many numeric variables. My experience has been that COMPRESS=YES reduces the size of the SAS data set by about 50 percent.

An option to use with COMPRESS= is REUSE=. Specifying this option allows SAS to reuse space within the compressed SAS data set that has been freed by deleted observations. Otherwise, SAS cannot reclaim the space made available by deleted observations.

Consider the following examples.

```
/*USE THE DATA STEP COMPRESS OPTION*/
DATA TEMP.FILE2(COMPRESS=BINARY
    REUSE=YES);
  SET WORK.FILE1;
  WHERE REC_TYPE='1';
RUN;
/*USE THE SYSTEM COMPRESS OPTION*/
OPTIONS COMPRESS=YES REUSE=YES;
DATA WORK.FILE3;
  SET WORK.FILE1;
  WHERE REC_TYPE='1';
RUN;
```

After running the DATA step with the COMPRESS= set to “YES” or “BINARY” you will see a message in your SAS log that looks something like this:

```
NOTE: The data set WORK.FILE3 has 117658
observations and 20 variables.
NOTE: Compressing data set WORK.FILE3
decreased size by 31.70 percent.
Compressed is 935 pages; un-compressed
would require 1369 pages.
```

I thought you might want to see some benchmark results. I took four SAS data sets with differing numbers of observations and variables. I started with uncompressed SAS data sets, then compressed with normal compression, then compressed with binary compression. File1 and File3 have 20 variables, 5 of which are numeric; and File2 and File4 have 29 variables, 10 of which are numeric.

				COMPRESS REDUCTION	
	Obs	Char	Num	YES	BINARY
File1	120,641	15	5	31.91%	23.72%
File2	183,476	19	10	52.10%	49.22%
File3	1,542,257	15	5	31.38%	21.02%
File4	6,976,838	19	10	48.81%	46.61%

In my examples, COMPRESS=YES always produced better results. However, the files with more variables and more numeric variables (File2 and File4) got almost the same benefit using

COMPRESS=BINARY. I recommend you try both methods with your files to determine which provides the better performance.

In some cases, compressing a SAS data set will result in a file that is larger in storage size than the uncompressed original. When this happens, SAS will warn you with a message. Version 8 of the SAS System will not compress the SAS data set when the result would be a larger file size.

Now take a look at the space savings (in this example, from MS-Windows).

Name	Size	Type
file1_binary.sas7bdat	8,569KB	SAS System Data Set
file1_compress.sas7bdat	7,649KB	SAS System Data Set
file1_uncompress.sas7bdat	11,497KB	SAS System Data Set
file2_binary.sas7bdat	13,809KB	SAS System Data Set
file2_compress.sas7bdat	13,025KB	SAS System Data Set
file2_uncompress.sas7bdat	27,529KB	SAS System Data Set
file3_binary.sas7bdat	113,321KB	SAS System Data Set
file3_compress.sas7bdat	98,457KB	SAS System Data Set
file3_uncompress.sas7bdat	146,889KB	SAS System Data Set
file4_compress.sas7bdat	529,153KB	SAS System Data Set
file4_uncompress.sas7bdat	1,046,533KB	SAS System Data Set
file4_binary.sas7bdat	551,833KB	SAS System Data Set

As you can see, the space savings are well worth the effort. In the case of File4, a reduction of nearly 500MB is nothing to gloss over.

DELETE UNNEEDED VARIABLES

Deleting unneeded variables can have a dramatic impact on the size of the SAS data set. For example, a variable of only five bytes in a SAS data set of one million observations will require five million bytes, or approximately 5MB. As soon as possible in your DATA steps and procedures, delete any SAS data set variables that you do not need. Use the data set DROP= option to identify which variables to delete, or use the KEEP= option to identify which variables to retain. Both will accomplish the same thing, one will be easier to use than the other depending the number of the existing variables you want to eliminate.

```
/*DROP UNNEEDED VARIABLES*/
PROC SORT DATA=DISK.WIP_DTL
    (DROP=TTDOTAMT TTDOTHRS TTDTOAMT
          TTDTOHRS TTDSRAMT)
     OUT=DISK.SORTED(DROP=REC_TYPE);
  BY POOL ACCOUNT DEPT DATE;
  WHERE REC_TYPE NE ' ';
RUN;
/*SUMMARIZE SORTED FILE*/
PROC SUMMARY DATA=DISK.SORTED MISSING;
  BY POOL ACCOUNT DEPT DATE;
  VAR YTDOTHRS YTDTOHRS YTDOTAMT
      YTDTOAMT YTDSRAMT;
  OUTPUT OUT=DISK.WIP_SUM
    (INDEX=(PBCODE) DROP=_TYPE_)
    SUM=YTDOTHRS YTDTOHRS YTDOTAMT
        YTDTOAMT;
RUN;
```

Frequently, when creating a subset or summary SAS data set or just processing a SAS data set with a procedure, you do not need all of the variables in the source SAS data set. So delete those you do not need from the source SAS data set, and do so as soon as possible. However, be careful not to delete any variables that you will need. Many times I have fallen into the trap of dropping a variable from my input SAS data set and then referred to it in a WHERE statement. The solution, of course, is to keep the variable in the input SAS data set and drop it from the output SAS data set. Consider the example at the bottom of the previous column.

Notice in the SORT procedure that we do not drop the REC_TYPE variable from the input (DATA=) SAS data set because we need it for the WHERE statement. If we had included it in the DROP= option on the input SAS data set, we would have received an error that the REC_TYPE variable was not on the input SAS data set. In this example, the SUMMARY procedure will produce a SAS data set with only the character variables identified in the BY statements and the four numeric variables identified on the VAR statement (plus, of course, the _FREQ_ and _TYPE_ variables created by the SUMMARY procedure, unless we drop them). So, why bother dropping the unwanted variables during the SORT procedure? Because by dropping those variables in the SORT procedure, our intermediate WORK SAS data set is greatly reduced in size. (And, we will also see a reduction in the time and space needed for sorting.)

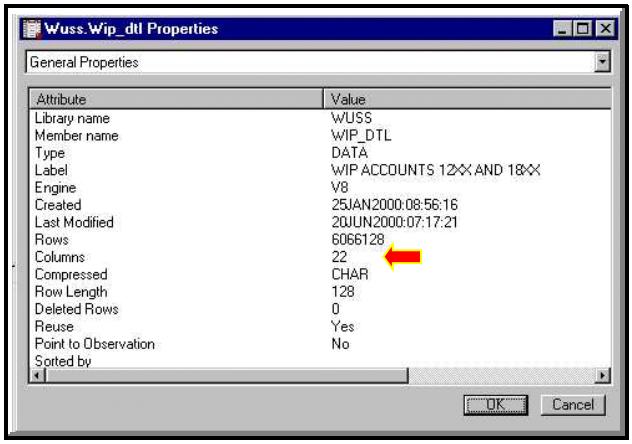
Name	Size	Type
ind_dtl.sas7bdat	89,161KB	SAS System Data Set
sorted.sas7bdat	309,969KB	SAS System Data Set
wip_dtl.sas7bdat	391,833KB	SAS System Data Set
wip_sum.sas7bdat	81KB	SAS System Data Set
wip_sum.sas7bndx	29KB	SAS System Data S...

Let's take a look at the WIP_DTL and the SORTED file from Windows Explorer's point of view. Dropping five variables made a significant difference.

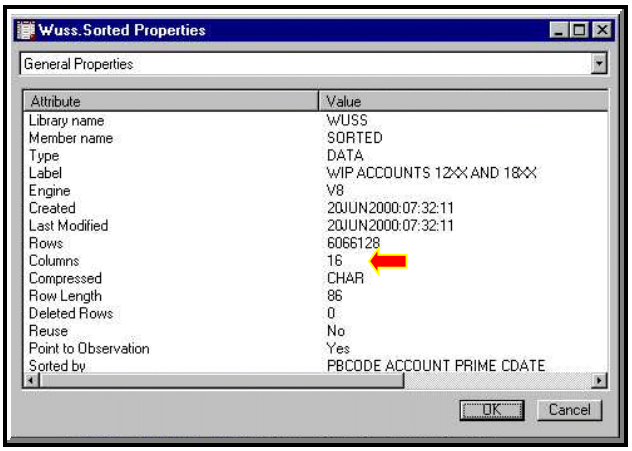
Now look at the file properties from SAS' point of view. First notice the SAS data set WIP_DTL. It contains 22 variables.

Do Not Overlook the Obvious

Delete unneeded variables as soon as possible. Even a two character variable can add up to a significant amount of space if you have many observations.



Now, notice the SAS data set SORTED. It contains only 16 variables.



Deleting unneeded variables is such a great way to reduce space, I really want to emphasize this trick. I really want to emphasize this trick. When you reduce space, you can also reduce processing time and I/O time. Let's look at an example of sorting a SAS data set. First, we will sort it without dropping any of the unneeded variables. Second, we will drop the unneeded variables, but will we do so using a KEEP= option on the output SAS data set. Third, we will drop the unneeded variables, using a KEEP option on the input SAS data set. Look at the SAS log carefully, noticing the time and memory differences.

Here's the SAS code for our first scenario, keeping everything.

```
/*KEEP ALL VARIABLES*/
PROC SORT DATA=SUGI.QUARTER1
          OUT=WORK.SORTED1A;
  BY WEEK PROJECT DEPT ACCOUNT;
RUN;
```

And, here's the SAS log for our first scenario.

```
NOTE: There were 167184 observations
read from the data set SUGI.QUARTER1.
NOTE: The data set WORK.SORTED1A has
167184 observations and 24 variables.
NOTE: PROCEDURE SORT used:
      real time           1:11.01
      Memory              2130k
```

Here's the SAS code for our second scenario. Notice the KEEP= option on the output SAS data set. In this scenario, we will be sorting all of the variables from the input SAS data set, but not writing all of them to the output SAS data set.

```
/*DROP VARIABLES ON OUTPUT*/
PROC SORT DATA=SUGI.QUARTER1
          OUT=WORK.SORTED1B
    (KEEP=WEEK PROJECT DEPT ACCOUNT
    MTDTOAMT MTDOTAMT MTDSRAMT);
  BY WEEK PROJECT DEPT ACCOUNT;
RUN;
```

Now, here's the SAS log from our second scenario. You will notice some improvement in processing time over our first scenario, but no memory improvement.

```
NOTE: There were 167184 observations
read from the data set SUGI.QUARTER1.
NOTE: The data set WORK.SORTED1B has
167184 observations and 7 variables.
NOTE: PROCEDURE SORT used:
      real time           50.52 seconds
      Memory              2123k
```

Now, here's the SAS code for our third scenario. This time, we're being smart and dropping all of the unneeded variables from the input SAS data set.

```
/*DROP VARIABLES ON INPUT*/
 PROC SORT DATA=SUGI.QUARTER1
    (KEEP=WEEK PROJECT DEPT ACCOUNT
    MTDTOAMT MTDOTAMT MTDSRAMT)
    OUT=WORK.SORTED1C;
  BY WEEK PROJECT DEPT ACCOUNT;
RUN;
```

Finally, take a look at the SA Slog from our third scenario.

```
NOTE: There were 167184 observations
read from the data set SUGI.QUARTER1.
NOTE: The data set WORK.SORTED1C has
167184 observations and 7 variables.
NOTE: PROCEDURE SORT used:
      real time           14.56 seconds
      Memory              2236k
```

We do not see much difference in memory usage, but, wow, look at the processing reduction! In our first scenario, processing time

was over 1 minute. In our second scenario, processing time was over 50 seconds. But, by dropping the unneeded variables from the input SAS data set, processing time went down to just over 14 seconds! Just imagine what you can do with those extra 36 seconds! But, seriously, the important thing to note here is the percentage change. If the original sort had taken several minutes, the reduction in time would be very noticeable. So, in addition to reducing sort space, you can greatly reduce processing time as a result of the reduced space.

Of course, when you drop unneeded variables from the input SAS data set, you don't affect the saved input SAS data set. SAS will just drop the specified variables as it reads the input SAS data set into the Program Data Vector.

DELETE UNNEEDED OBSERVATIONS

Any observations that are not needed in your temporary or permanent SAS data sets just take up space. So, delete them as soon as possible. Do this with a WHERE statement when processing SAS data sets in a DATA step or procedure, or an IF statement when processing external files in a DATA step. Consider your data needs for the SAS data sets you will create in your temporary work space and permanent storage space. If you will not need all of the observations in the SAS data set, get rid of those you do not need from your input SAS data set. Consider the sample SORT procedure in the above section as an example of using the WHERE statement to delete unneeded observations.

REDUCE THE NUMBER OF SAS DATA SETS

There are a number of strategies to use to reduce the number of SAS data sets processed and stored within the same temporary and permanent storage space. Let's look at my favorite strategies.

OUTPUT SORTED SAS DATA SETS

When you sort a SAS data set using the SORT procedure and you do not specify an output SAS data set using the OUT= option, SAS will overwrite the input SAS data set with the sorted SAS data set. To do this, SAS must create a temporary sorted SAS data set until the SORT procedure successfully completes. Then SAS will overwrite the old input SAS data set with the temporary sorted SAS data set. This method works, of course, but creates a temporary sorted file in the WORK library. If you are short on WORK space, this may be a problem. When you specify an output file for the sorted SAS data set you will still have two SAS data sets created: the input SAS data set and the sorted output SAS data set. However, in this situation you can place the sorted SAS data set somewhere other than the WORK library, such as another temporary SAS data library, a tape SAS data library, or a permanent disk SAS data library. Consider the following example.

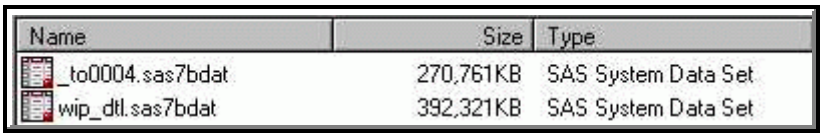
```
/*SORT TO OUT= DESTINATION*/
PROC SORT DATA=WORK.WIP_DTL
          OUT=TAPE.SORTED;
  BY PBCODE ACCOUNT PRIME CDATE;
RUN;
```

DELETE SAS DATA SETS BEFORE REPLACING THEM

Whenever you update or replace a SAS data set with new data, SAS will create a temporary SAS data set to hold the new data until the DATA step or procedure successfully completes. Then SAS will overwrite the old input SAS data set with the one just created. This causes SAS to create a temporary SAS data set in the WORK library (I noticed that MS-Windows places this temporary SAS data set in the same library as the permanent SAS data set you are creating). Let's look at a real-life example under MS-Windows.

```
/*OVERWRITE AN EXISTING SAS DATA SET*/
DATA DISK.WIP_DTL;
  SET TAPE.WIP_DTL;
  WHERE REC_TYPE NE '  ';
RUN;
```

If the WIP_DTL SAS data set already existed in the DISK library, the following would happen in the DISK library.



Notice the temporary file created because the target output SAS data set already existed. This, of course, is wasting space. Instead, delete the existing SAS data set using the DATASETS or the DELETE procedure before running the DATA step or procedure that will update or replace the SAS data set. This method cannot be used, of course, if the existing SAS data set is needed as the basis for updating or replacing the SAS data set. Consider the following example.

```
/*DELETE EXISTING FILE FIRST*/
PROC DATASETS LIBRARY=DISK NOLIST;
  DELETE WIP_DTL;
QUIT;
DATA DISK.WIP_DTL;
  SET TAPE.WIP_DTL;
  WHERE REC_TYPE NE '  ';
RUN;
```

USE AN ALTERNATE TEMPORARY LIBRARY

When you do not specify a library for a SAS data set, SAS will place the data set in the WORK library. Of course, you can also specify the WORK library in the two level SAS data set name to place the SAS data set in the WORK library. When you are short of space in the WORK library and need to have more than one SAS data set in temporary storage simultaneously, an alternate approach is to create alternate temporary libraries. Yes, you can do this. Creating an alternate temporary library is a very simple process. Just allocate a temporary SAS data library on a different space than your library. How to do this is operating system dependent. On a PC, you must allocate the alternate temporary library on a hard disk drive other than the disk drive where the WORK library is allocated. If you allocate the alternate temporary library on the same hard disk drive, then the temporary library you allocate and the WORK library will be competing for the same limited space.

Once you have allocated one or more alternate temporary libraries, use it for some SAS data sets that you need to place in temporary work space. For example (assuming your SAS WORK library is on the C: drive and you have a D: drive available):

```
LIBNAME TEMP
    "D:\LONG\WINDOWS\FOLDERNAME";
```

Then in your SAS program, you might do something like the following:

```
LIBNAME CD
    "D:\LONG\WINDOWS\FOLDERNAME\DATA";
/*SPLIT THE INPUT FILE TO MULTIPLE
STORAGE DESTINATIONS*/
DATA WORK.FILE1 TEMP.FILE2;
  SET CD.MASTER
  SELECT (DIVISION);
    WHEN ("01") OUTPUT WORK.FILE1;
    WHEN ("02") OUTPUT TEMP.FILE2;
    OTHERWISE DELETE;
  END;
RUN;
```

In this example, you are reading a permanent SAS data set stored on a CD and creating two temporary SAS data sets. The first temporary SAS data set is being written to the WORK library (WORK.FILE1) and the second temporary SAS data set is being written to the temporary library "TEMP" (TEMP.FILE2), which you have allocated on a hard disk drive other than the hard disk drive where SAS has allocated its WORK library.

CLEAN UP THE STORAGE SPACE

This tip is so obvious that overlooking it is easy. Simply delete SAS data sets in the WORK library, other temporary space, or permanent space when you no longer need them. Do this, of course, using the DATASETS or the DELETE procedure.

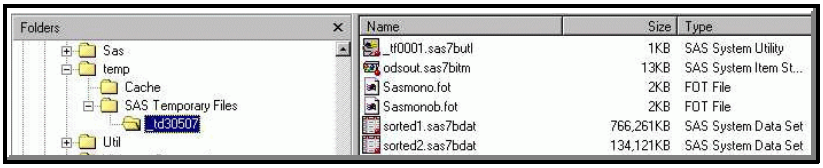
Consider the following typical job stream. Here you will sort a permanent SAS data set to a SAS data set in the WORK library that you will then summarize. Then, you will sort another permanent SAS data set to a SAS data set in the WORK library that you will then summarize. The first sorted SAS data set in the WORK library is not needed again after the first SUMMARY procedure has completed.

```
/*SORT THEN SUMMARIZE FIRST FILE*/
PROC SORT DATA=DISK.WIP_DTL
           OUT=WORK.SORTED1;
  BY POOL ACCOUNT DEPT DATE;
RUN:
PROC SUMMARY DATA=WORK.SORTED1 MISSING;
  BY POOL ACCOUNT DEPT DATE;
  VAR YTDTOAMT;
  OUTPUT OUT=DISK.WIP_SUM SUM=YTDTOAMT;
RUN;
/*SORT THEN SUMMARIZE SECOND FILE*/
PROC SORT DATA=DISK.IND_DTL
           OUT=WORK.SORTED2;
  BY POOL ACCOUNT DEPT DATE;
RUN:
PROC SUMMARY DATA=WORK.SORTED2 MISSING;
  BY POOL ACCOUNT DEPT DATE;
  VAR YTDTOAMT;
  OUTPUT OUT=DISK.IND_SUM SUM=YTDTOAMT;
RUN;
```

Do Not Overlook the Obvious

Do just like your mama told you...clean up after yourself.

We can see the result in the WORK library, in this example, from MS-Windows.



In this example, you have a very large file, SORTED1, that is just taking up space while you sort and summarize your second SAS data set. Take a look at the following alternative.

```
/*SORT THEN SUMMARIZE FIRST FILE*/
PROC SORT DATA=DISK.WIP_DTL
           OUT=WORK.SORTED1;
  BY POOL ACCOUNT DEPT DATE;
RUN:
PROC SUMMARY DATA=WORK.SORTED1 MISSING;
  BY POOL ACCOUNT DEPT DATE;
  VAR YTDTOAMT;
  OUTPUT OUT=DISK.WIP_SUM SUM=YTDTOAMT;
RUN;
/*DELETE USING PROC DATASETS*/
PROC DATASETS LIBRARY=WORK NOLIST;
  DELETE SORTED1;
QUIT;
RUN;
/*OR, DELETE USING PROC DELETE*/
/*
PROC DELETE DATA=WORK.SORTED1;
QUIT;
RUN;
*/
/*SORT THEN SUMMARIZE SECOND FILE*/ PROC
SORT DATA=DISK.IND_DTL
           OUT=WORK.SORTED2;
  BY POOL ACCOUNT DEPT DATE;
RUN:
PROC SUMMARY DATA=WORK.SORTED2 MISSING;
  BY POOL ACCOUNT DEPT DATE;
  VAR YTDTOAMT;
  OUTPUT OUT=DISK.IND_SUM SUM=YTDTOAMT;
RUN;
```

Instead, you can insert a DATASETS or DELETE procedure after summarizing the first SAS data set to delete the unneeded file. (Never heard of Proc DELETE? Neither had I until someone at SUGI 25 pointed it out to me. It has not been documented since version 5, but still works.)

CONCLUSION

SAS is such a powerful tool and functions so well with the operating systems on which it runs (most everything except a Commodore 64 and a TRSH 80) that there are ways around every problem. The problems associated with limited work space and limited permanent storage space can be resolved with a little resourcefulness.

ACKNOWLEDGMENTS

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. ® indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.

CONTACT INFORMATION

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

Curtis A. Smith
Defense Contract Audit Agency
P.O. Box 20044
Fountain Valley, CA 92728-0044
Work Phone: 714-896-4277
Fax: 413-383-6395
Email: casmith@mindspring.com