Paper 291-25

By Hook or By Crook: Overcoming Resource Limitations

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

ABSTRACT

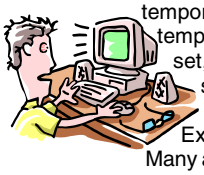
Have you ever had trouble getting a SAS job to complete, although your SAS code is bug free? Often, especially when processing large amounts of data, your operating environment just does not have enough resources to complete your SAS job the way you have written it. Likely, a solution to the resource problems you have or might encounter exists. Herein, the author will share some of his favorite tricks to squeeze more out of the operating environment and for working SAS code around resource limitations. These tricks will include SAS code, SAS options, and operating system parameters. This paper will cover subjects such as reducing the number of SAS data sets needed, reducing the size of SAS data sets, sorting efficiencies, optimizing data library parameters, and cleaning up the work environment.

INTRODUCTION

In my endeavors to get my large SAS jobs to complete, I encounter four categories of problems: temporary work space shortages, permanent storage space shortages, memory shortages, and (uh, I can't remember the other one - oh, yeah) miscellaneous stuff. So, I will approach the problems I encounter and the solutions I use in these four groupings. Some of the problems and solutions presented are unique to IBM MVS, some are unique to MS-Windows, but most are common to all operating systems.

TEMPORARY WORK 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. For example, if you create a SAS data set to replace an existing permanent SAS data set, SAS will first create a temporary SAS data set. Then SAS will move the temporary SAS data set over the existing SAS data set, but only after the DATA step has completed successfully.



Exhausting temporary storage space is common. Many approaches to solve the problem of out of work space are available. These fall into at least four categories: increase the size and optimize the temporary work space; reduce the size of the data sets being stored in temporary work space; decrease the number of data sets processed in the temporary work space; and clean up the temporary work space during the SAS job.

Increase the Size of and Optimize the WORK Library

When you launch SAS it will create its WORK library using the parameters provided to it. You can increase the size of the WORK library and alter the parameters of the WORK library to make the WORK library larger and more efficient.

IBM MVS WORK Library Space Under IBM MVS you can increase the size of the space (SPACE) parameter of the physical file created for the WORK library. If you are running SAS as a batch job, you can change the SPACE parameters, in the DD statement for the WORK data set. Simply increase the primary and/or secondary allocations and specify the desired allocation units. For example,

```
//WORK  DD UNIT=SYSDA,SPACE=(CYL,(100,50)),
//         DCB=(RECFM=FS,DSORG=PS,
//         LRECL=27648,BLKSIZE=27648)
```

Here, we are allocating cylinders, with 100 primary and 50 secondary. If you use a procedure in your JCL to run SAS you may need to change your JCL to run the SAS program so that you can override the DD statements used in the procedure. An example of JCL running the SAS program is shown below. Note the first line, where the SAS program is run.

```
//SASJOB    EXEC PGM=SASHOST,TIME=299,
//               PARM=('OPLIST FULLSTIMER')
//STEPLIB   DD DSN=SUSP.SAS.LIBRARY,DISP=SHR
//CONFIG    DD DSN=SUSP.SAS.CNTL(BATCHXA),DISP=SHR
//         DD DSN=NULLFILE,DISP=SHR
//CTRANS    DD DSN=SUSP.SAS.SASC.TRANSLIB,DISP=SHR
//SASAUTOS  DD DSN=SUSP.SAS.AUTOLIB,DISP=SHR
//SASHELP   DD DSN=SUSP.SAS.SASHELP,DISP=SHR
//SASMSG    DD DSN=SUSP.SAS.SASMSG,DISP=SHR
//WORK      DD UNIT=SYSDA,SPACE=(CYL,(100,50)),
//            DCB=(RECFM=FS,DSORG=PS,LRECL=27648,
//                 BLKSIZE=27648)
```

An alternative is to pass the SPACE parameter to SAS on the EXEC card and eliminate the WORK DD statement. Do this as follows:

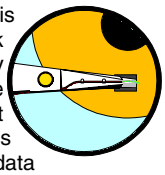
```
//SASJOB    EXEC PGM=SASHOST,WORK='10000,50',TIME=299
```

In this case, the parameters passed are blocks. To control all of the DCB parameters for the WORK library, I recommend using the DD statement for the WORK library rather than just passing the SPACE parameter.

When running SAS interactively, specify the size of the WORK library using the desired SPACE parameter for the desired amount of blocks. Accomplish this with an option on the SAS invocation command. For example,

```
SAS work ('10000,50')
```

IBM MVS WORK Library Block Size A great way to save space is to tune the block size (BLKSIZE) parameter associated with the WORK library allocation. The block size is set when you create a physical file. This relates to the number of observations transferred together as a group. Each block is separated by a marker, known as the interblock gap, that occupies space but does not hold any data. The smaller the block size, the more interblock gaps that will exist in the same amount of physical space. Therefore, smaller block sizes result in more interblock gaps in your SAS data libraries, and thus, more unusable space. Also, larger block sizes transfer more observations as a group and can increase I/O throughput (decreasing EXCPs). To save space in the WORK library and improve performance, larger SAS data sets work better with a large block size, like 27648 as in the example above. Set the LRECL parameter to match the block size or ignore the LRECL parameter and let SAS set it for you. You can read a thorough and technical discussion of block sizes in Michael A. Raithel's book "Tuning SAS Applications in the MVS Environment," which is available through the SAS Institute.



IBM MVS Sort Space Sequential SAS data libraries do not store observation information in the header of the SAS data library. This means that SAS does not know how many observations are in a SAS data set stored on tape. This can create some problems. When you sort a SAS data set (SORT, MEANS, SUMMARY procedures, etc.) SAS will allocate the sort work data sets it needs, determine their size, pass the SIZE parameter to the host sort utility, and choose the sort program (unless these options are overridden with an OPTIONS statement). SAS chooses the values for these

options based upon the number of observations in the SAS data set. However, when the SAS data set is on tape, SAS does not know the number of observations. Thus, SAS cannot properly set the needed sort parameters. It is very easy for a sort routine to fail for lack of sort space when your SAS data sets are stored on tape. SAS will return the following message:

```
Sort did not complete successfully. See messages
on the Job Console Log or //SYSOUT data set.
```

To overcome this, specify the following sort options with an OPTIONS statement.

SORTPGM= This option tells SAS to use the host sort utility or the SAS sort program. Typically, the host sort utility, such as DFSORT, SYNCSORT, or DJSORT, will do better on large SAS data sets. Select either 'SAS' for the SAS sort utility or 'HOST' for the host sort utility.

SORTWKNO= This parameter tells SAS how many sort work data sets to use. This is a numeric value from 1-6.

SORT= This parameter specifies the minimum size of all sort work data sets. This is a numeric value for the number of cylinders. SAS will divide the size you select by the number of sort work data sets to figure out the size of each sort work data set.

SORTSIZE= This passes a value to the host sort utility for its SIZE parameter. This is a numeric value.

Here is an example:

```
OPTIONS  SORTPGM=HOST  SORTSIZE=256K  SORTWKNO=6
         SORT=2400;
```

This will cause SAS to use the host sort utility, with 256K for its SIZE parameter, and allocate six sort work data sets of 400 cylinders each. The amount of space SAS needs for sorting is a bit larger than the amount of space the SAS data set would occupy if it were stored on disk. Trial and error may be necessary to find the right values to use.

Really Big Files on MVS Sometimes your files may be so large that the limit of six sort workers that SAS can allocate just isn't enough because you may not be able to allocate enough cylinders in the SORT= parameter. It might be very difficult to get a large enough allocation on with only six sort workers. For example, you may not be able to allocate 6000 cylinders for your 6 sort workers because the operating system may not be able ti find that much available contiguous space. Fortunately, there is a solution. By allocating the sort workers externally by the host and using the host sort routine you can allocate as many as ninety-nine sort workers. Here's how to do it in your JCL with a DD statement.

```
//SASSWK01  DD UNIT=SYSDA,SPACE=(CYL,(800,50))
//SASSWK02  DD UNIT=SYSDA,SPACE=(CYL,(800,50))
//SASSWK03  DD UNIT=SYSDA,SPACE=(CYL,(800,50))
//SASSWK04  DD UNIT=SYSDA,SPACE=(CYL,(800,50))
//SASSWK05  DD UNIT=SYSDA,SPACE=(CYL,(800,50))
//SASSWK06  DD UNIT=SYSDA,SPACE=(CYL,(800,50))
//SASSWK07  DD UNIT=SYSDA,SPACE=(CYL,(800,50))
//SASSWK08  DD UNIT=SYSDA,SPACE=(CYL,(800,50))
```

The DD name prefix can be determined by checking with your site administrator or by looking at the output JES messages from a previous SAS job. The SAS default is 'SASS'. This prefix can also be set using the SORTWKDD= option. In this example, I have specified eight sort workers with 800 primary cylinders each. You will probably find that it is easier to get the total allocation space you need if you allocate more sort workers with smaller cylinder amounts rather than fewer sort workers with larger cylinder amounts.

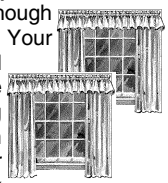
Next, within your SAS code specify the SAS option "DYNALLOC."

This option tells SAS to let the host sort utility dynamically allocate the sort workers, rather than having SAS allocate its maximum six sort workers. For example:

```
OPTIONS DYNALLOC SORTWKDD=SASS;
```

To identify and overcome other problems associated with SAS data sets stored on tape, see the author's paper entitled Data Libraries on Tape in the proceedings from the WUSS '98 conference and the SUGI 24 conference.

MS-Windows WORK Library Space Under MS-Windows or other microcomputer operating systems there are ways to increase the size of the WORK library and optimize it even though it resides on your microcomputer's hard disk. Your WORK library will be limited to the available hard drive space. Within your SAS configuration file (e.g., config.sas under version 6, SASv7.cfg under version 7, SASv8.cfg under version 8) is an option that selects the hard drive and folder for the SAS WORK library. The option will look something like this:



```
/* Setup the default SAS System user work folder */
-WORK "c:\TEMP\SAS Temporary Files\"
```

You can change the drive and folder of the SAS WORK library. One way to increase the size of the WORK library is to direct SAS to place the WORK library on a hard drive with sufficient space. If your primary hard drive does not have as much free space as your SAS jobs require for WORK space, place the WORK library on another local hard drive or on a network drive. Also, if you have more than one hard drive with enough space for your WORK library choose the fastest hard drive to improve performance (you would want to do the same for your MS-Windows virtual memory).

Another way to optimize your WORK library is to keep your hard drive cleaned and optimized. For example, delete unnecessary files from the hard drive. If your version of MS-Windows operating system support's FAT32 and yet your hard drives are configured as FAT16, changed them to FAT32. Doing so can recover as much as 40 percent of the hard drive. Windows 98 comes with a utility to convert a FAT16 hard drive to FAT32. Partition Magic will safely convert a FAT16 hard drive to FAT32 as will some freeware products. Also, use utilities such as ScanDisk, Dfrag, Norton Utilities, and SpinRite to keep the hard drive performing as efficiently and error free as possible.

Reduce the Size of the Data Sets in the WORK Library

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

Compressing Files Under some operating systems, including IBM MVS and MS-Windows, SAS can use compression algorithms to compress SAS data sets. 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. If those SAS data sets are placed in the WORK library, you can use the COMPRESS= option to extend the WORK library space. To use the COMPRESS= system or data set option, set the option to either "YES" or "BINARY." The COMPRESS=YES value uses an algorithm that works better with SAS data sets that primarily have character variables. On the other hand, COMPRESS=BINARY uses a different algorithm that works better with SAS data sets that have many variables including many numeric variables. My experience has been that COMPRESS=YES results in about 50 percent less storage space.

Another 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 deleting an observation. Otherwise, SAS cannot reclaim the deleted space.

Consider the following examples.

```
DATA TEMP.FILE2(COMPRESS=BINARY REUSE=YES);
        SET WORK.FILE1;
        WHERE REC_TYPE='1';
RUN;

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 note 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.
```

Delete Unneeded Variables 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.

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. So delete those you do not need from the source SAS data set. However, be sure not to delete any variables that you need to process within the DATA step or procedure. For example, if you need to perform a WHERE selection on a variable but do not want that variable in the output SAS data set you will need to keep it on the input SAS data set but can drop it from the output SAS data set. 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. Consider the following example.

```
PROC SORT DATA=TAPE.WIP_DTL
        (DROP=MTDOTAMT MTDOTHRS MTDTOAMT MTDTOHRS)
         OUT=WORK.SORTED(DROP=REC_TYPE);
  BY PBCODE ACCOUNT PRIME CDATE;
  WHERE REC_TYPE='1';
RUN;

PROC SUMMARY DATA=WORK.WIP_DTL MISSING;
  BY PBCODE ACCOUNT PRIME CDATE;
  VAR YTDOTHRS YTDTOHRS YTDOTAMT YTDTOAMT;
  OUTPUT OUT=DISK.WIP_SUM
          (INDEX=(PBCODE) DROP=_TYPE_)
         SUM=YTDOTHRS YTDTOHRS YTDOTAMT YTDTOAMT;
RUN;
```

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, 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, our intermediate WORK SAS data set is greatly reduced in size.

Delete Unneeded Observations Any observations that are not needed in your temporary 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. If you will not need all of the observations in the SAS data set, get rid of those you do not need. Consider the sample DATA step in the above section.

Decrease the Number of Data Sets Processed in WORK

There are several ways you can reduce the number of data sets SAS will need to put into the WORK library. Below are my favorite strategies for reducing the number of SAS data sets in the WORK library.

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 WORK and need to have more than one SAS data set in temporary storage simultaneously an approach is to create additional temporary work libraries. Below are ways to do this under IBM/MVS and under MS-Windows.

IBM/MVS Creating an alternate temporary work library is a very simple process. Just allocate a temporary SAS data library and use it for some SAS data sets that you need to place in temporary work space. For example,

```
LIBNAME TEMP "LONG.MVS.FILENAME.TEMPLIB"
        DISP=(NEW,DELETE,DELETE) UNIT=SYSDA
        SPACE=(CYL,(200,25)) BLKSIZE=27648;
```

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

```
LIBNAME TBLS "LONG.MVS.FILENAME.TABLES" DISP=SHR;
FILENAME RAWDATA TAPE "LONG.MVS.FILENAME.RAWDATA"
         DISP=OLD;
RUN;

DATA WORK.FILE1 TEMP.FILE2 WORK.DISCARD;
  SET TBLS.RAWDATA
  SELECT (DIVISION);
    WHEN ("AC") OUTPUT WORK.FILE1;
    WHEN ("CG") OUTPUT TEMP.FILE2;
    OTHERWISE   OUTPUT WORK.DISCARD;
  END;
RUN;
```

If you need more than one additional temporary work library, simply allocate as many as you need. If you need to put some really big files in a temporary work library and cannot get enough disk space, allocate the temporary work libraries to tape. Generally, you will want to do this in the JCL as follows.

```
//TEMPTAPE  DD DSN=LONG.MVS.FILENAME.TEMPTAPE,
//             DISP=(NEW,PASS),
//             DCB=(RECFM=FS,DSORG=PS,LRECL=27648,
                   BLKSIZE=27648)
```

MS-Windows Creating an alternate temporary work library is a very simple process. Just allocate a temporary SAS data library on a different hard drive than your SAS WORK library. If you use the same hard disk drives, then the temporary library you allocate and the WORK library will be competing for the same limited space. Then 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 TBLS "C:\LONG\WINDOWS\FOLDERNAME\TABLES";
FILENAME RAWDATA
     "C:\LONG\WINDOWS\FOLDERNAME\RAWDATA.DAT";
DATA WORK.FILE1 TEMP.FILE2 WORK.DISCARD;
  SET TBLS.RAWDATA
  SELECT (DIVISION);
    WHEN ("AC") OUTPUT WORK.FILE1;
    WHEN ("CG") OUTPUT TEMP.FILE2;
    OTHERWISE   OUTPUT WORK.DISCARD;
  END;
RUN;
```

Output Sorted Sas Data Sets When you sort a SAS data set using the SORT procedure if 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 input SAS data set with the sorted SAS data set. This method works, of course, but creates a temporary sorted file in the WORK library. If 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.

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

Delete Permanent SAS Data Sets Before Replacing Them Whenever you update or otherwise replace a permanent SAS data set, 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 SAS data set with the one just created. This causes SAS to create a temporary SAS data set in the WORK library. Instead, delete the existing SAS data set using the DATASETS 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.

```
LIBNAME &CC.19&FY.
    "LONG.MVS.FILENAME.&CC.19&FY.";
PROC DATASETS LIBRARY=&CC.19&FY. NOLIST;
  DELETE WIP_DTL;
QUIT;
RUN;
DATA &CC.19&FY..WIP_DTL
     (LABEL='ACCOUNTS 12XX AND 18XX');
  SET TAPE&CC.&FY..WIP_MON;
  WHERE REC_TYPE='1';
RUN;
```

Clean up the WORK Library

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

```
PROC SORT DATA=TAPE.WIP_DTL OUT=WORK.SORTED1;
  BY PBCODE ACCOUNT PRIME CDATE;
PROC SUMMARY DATA=WORK.SORTED1 MISSING;
  BY PBCODE ACCOUNT PRIME CDATE;
  VAR YTDTOAMT;
  OUTPUT OUT=&CC.19&FY..WIP_SUM SUM=YTDTOAMT;
RUN;
PROC DATASETS LIBRARY=WORK NOLIST;
  DELETE SORTED1;
QUIT;
RUN;
```

```
PROC SORT DATA=TAPE.IND_DTL OUT=WORK.SORTED2;
  BY PBCODE ACCOUNT PRIME CDATE;
PROC SUMMARY DATA=WORK.SORTED2 MISSING;
  BY PBCODE ACCOUNT PRIME CDATE;
  VAR YTDTOAMT;
  OUTPUT OUT=&CC.19&FY..IND_SUM SUM=YTDTOAMT;
RUN;
```

PERMANENT STORAGE SPACE SHORTAGES

Most of the tips mentioned earlier for reducing space in temporary work space will also work for reducing space in permanent storage. I do have a couple additional strategies for improving permanent storage.

Use Removable Storage Devices

Yes, not all permanent SAS data sets need to be stored on permanent disk drives. Under IBM MVS and other operating systems you can use tape devices. Limitations imposed by tape devices exist because they are sequential storage devices, but they work very well for storing very large SAS data sets. To identify and overcome problems associated with SAS data sets stored on tape, see the author's paper entitled Data Libraries on Tape in the proceedings from the WUSS '98 conference and the SUGI 24 conference.

Under MS-Windows and other microcomputer operating systems you can store large SAS data sets on removable storage devices such as ZIP drives, Jazz drives, and writeable CDs. While you might need to create the permanent SAS data sets on permanent disk and then move them to removable media using imaging software, this strategy is workable.

Create Summary Files

If you need all of the data in a large SAS data set but do not need all of the detail you can create summary SAS data sets from the detail file and keep only the summary SAS data sets on your permanent disk. For example, suppose you have a SAS data set with forty character variables and six numeric variables and you typically only need ten of the character variables with all the numeric values. Simply use the SUMMARY procedure to summarize the detail SAS data set by the ten character variables and sum the numeric variables. Place the summarized SAS data set in a permanent SAS data library on disk and keep the detail SAS data set in a permanent SAS data set on tape or other removable storage device.

MEMORY SHORTAGES

There was something I wanted to say about memory shortages, but I can't remember . . . oh yeah. Frequently, you may either find your job failing for lack of memory or find your job running slowly because the operating system is forced to use virtual memory rather than physical memory. While adding more physical memory always helps, this may not be an option. There are other strategies to use to help manage memory.

MVS Region Size

When running SAS in batch mode under IBM MVS you can increase the size of the REGION parameter to allow more memory for your job. You can set this in your JCL on the job card. Below is an example.

```
//MYJOB   JOB (W007285,,'JAAOBBHCCNDD316X'),
// 'SAS IS COOL     J316',REGION=7680K,TIME=600
```

Check with your site administrator to find valid values for the REGION parameter.

MS-Windows Memory

MS-Windows and MS-Windows applications are not very good about using memory and cleaning up after themselves. Many programs load large libraries into memory although you may not need all the functions stored in the libraries. Some programs do not unload everything from memory when you exit the program. With all this junk in physical memory, other programs and data, like SAS and your SAS data sets, may find themselves loaded into virtual memory (which, of course, is much slower then physical memory).

SAS users have a cheap solution to this problem. Shareware and freeware programs are available that routinely flush unneeded stuff from physical memory and move them to virtual memory. Doing so allows your SAS program to load itself and your SAS data sets into physical memory, thus improving performance. My two favorite programs are RamBooster and FreeMem. I have the free 'light' version of FreeMem, called FreeMem Standard. This version does not have all of the features of the shareware version, called FreeMem Professional. RamBooster is free and I find it better than FreeMem Standard. However, I do not know how it compares to FreeMem Professional. You can find RamBooster at http://www.sci.fi/~borg/rambooster/index.htm and FreeMem at http://www.3bsoftware.com.

Sort Before Using the SUMMARY Procedure

When using the SUMMARY procedure you can specify the summary variables with the BY or the CLASS statements. Within the SUMMARY procedure the BY statement requires that the input SAS data set must already be sorted or indexed in the same sequence as you specify in the BY statement. In contrast, within the SUMMARY procedure the CLASS statement will sort the observations in the input SAS data set according to the variables you specify in the CLASS statement. Is there a difference? Well, using the CLASS statement saves you from running a SORT procedure. However, SAS must still sort the input SAS data set. It is just going to do it within the SUMMARY procedure. And it will do this in the WORK library. I have found that using the CLASS statement running under IBM MVS with large SAS data sets takes longer than using the BY statement, including the time necessary to use the SORT procedure. I have also found that using the CLASS statement running under IBM MVS and MS-Windows with large SAS data sets takes much more memory than using the BY statement, including the memory necessary to use the SORT procedure.

And another thing, if you use a SORT procedure and then the SUMMARY procedure with the BY statement, you can direct the sorted SAS data set to a temporary library other than the WORK library (see the discussion above). Consider the example below.

```
PROC SORT DATA=WORK.WIP_DTL OUT=TEMP.SORTED;
  BY PBCODE ACCOUNT PRIME CDATE;
PROC SUMMARY DATA=TEMP.SORTED MISSING;
  BY PBCODE ACCOUNT PRIME CDATE;
  VAR YTDOTHRS YTDTOHRS YTDOTAMT YTDTOAMT;
  OUTPUT OUT=&CC.19&FY..WIP_SUM
     (INDEX=(PBCODE) DROP=_TYPE_)
     SUM=YTDOTHRS YTDTOHRS YTDOTAMT YTDTOAMT;
RUN;
```

For those of you who like benchmarking these things, I did some benchmarking. However, I do not have the space here to present the data, only tell you of my results. Under IBM MVS I ran a SORT procedure then a SUMMARY procedure using the BY statement and compared to running a SUMMARY procedure with the CLASS statement. I did two comparisons: one with a SAS data set containing 33,781 observations and the other with a SAS data set containing 133,162 observations. As noted above, the most dramatic difference seems to be that larger SAS data sets require the SUMMARY procedure to use far more memory when using the CLASS statement than when using the BY statement.

MISCELLANEOUS STUFF

There are always a few miscellaneous things that do not fit into any other groups: so here they are.

MVS TIME Parameter

It is really a bummer when your MVS SAS job stops suddenly only because you did not allocate enough CPU time to the job. Within the JCL for your batch job you may need to adjust the time for the SAS program and/or you may need to adjust the time for the entire batch job. You will find the TIME parameter for the entire job on the JCL job card. And you will find the TIME parameter for the SAS task, or step on the EXEC card. Look at the example below.

```
//MYJOB   JOB (W007285,,'JOHN316613NHOJX'),
// 'SAS IS COOL     4277',REGION=7680K,TIME=600
//SASSTEP   EXEC PGM=SASHOST,TIME=299,
            PARM=('OPLIST FULLSTIMER')
//STEPLIB   DD DSN=SUSP.SAS.LIBRARY,DISP=SHR
```

MVS Execution

Your site administrator probably has a handy-dandy SAS procedure for you to use to launch SAS in your JCL. However, if you run the procedure you may be stuck with some default values you do not want to use, such as the DCB parameters for the WORK library. Instead, execute the SAS program and specify the necessary DD statements, thereby taking control of the SAS invocation. Consider the following SAS JCL.

```
//SAS        EXEC PGM=SASHOST,TIME=299,
             PARM=('OPLIST FULLSTIMER')
//STEPLIB    DD DSN=SUSP.SAS.LIBRARY,DISP=SHR
//CONFIG     DD DSN=SUSP.SAS.CNTL(BATCHXA),DISP=SHR
//           DD DSN=NULLFILE,DISP=SHR
//CTRANS     DD DSN=SUSP.SAS.SASC.TRANSLIB,DISP=SHR
//SASAUTOS   DD DSN=SUSP.SAS.AUTOLIB,DISP=SHR
//SASHELP    DD DSN=SUSP.SAS.SASHELP,DISP=SHR
//SASMSG     DD DSN=SUSP.SAS.SASMSG,DISP=SHR
//WORK       DD UNIT=SYSDA,SPACE=(CYL,(100,50)),
//             DCB=(RECFM=FS,DSORG=PS,LRECL=27648,
                  BLKSIZE=27648)
```

I especially like to increase the default size of the WORK library and increase the block size to reduce the size of the interblock gaps. You can also specify a different location for the CONFIG file so you can change the default options set up at your site.

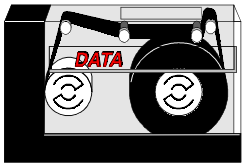
Eliminate the Need for Multiple Tape Devices

Your site has a finite number of tape devices. When you need multiple tape libraries in the same SAS job and if you are not careful and plan, your job will require the operating system to allocate as many tape drives as you specify tape libraries. You may be waiting awhile for the tape drives to become available. Consider the example of reading a huge SAS data set into a SAS DATA step where you want to split the file into five new SAS data sets. Each of the SAS data sets will be stored in a separate tape library. You specify each libref and SAS data set name on the DATA step like this:

```
data tape1.file tape2.file tape3.file tape4.file
     tape5.file;
 set tapein.input;
 more SAS statements
run;
```

Here, you will need six tape devices at once. Your job will wait, maybe for hours, for six drives to become available. Then, when your job grabs six tape drives, everyone at your site will wait for your job to release the drives. Not a very good plan. Instead, you can create the subsets one at a time. This will require only two tapes at once. You can write your code with a SAS macro like this:

```
%macro subset;
data &lib..file;
 set tapein.input;
 more SAS statements
run;
%mend;
%let lib=tape1;
%subset;
%let lib=tape2;
%subset;
etc.
```



To keep the operating system from still allocating all your tapes at once and on different tape drives, use the UNIT=AFF (AFFINITY) parameter in your JCL or SAS LIBNAME statement on the second through the last tape library to which you are writing. The JCL would look like this:

```
//TAPEIN   DD DSN=LONG.MVS.FILENAME.TAPEIN,DISP=OLD
//TAPE1    DD DSN=LONG.MVS.FILENAME.TAPE1,DISP=OLD
//TAPE2    DD DSN=LONG.MVS.FILENAME.TAPE2,DISP=OLD,
             UNIT=AFF=TAPE1
//TAPE3    DD DSN=LONG.MVS.FILENAME.TAPE3,DISP=OLD,
             UNIT=AFF=TAPE2
//TAPE4    DD DSN=LONG.MVS.FILENAME.TAPE4,DISP=OLD,
             UNIT=AFF=TAPE3
//TAPE5    DD DSN=LONG.MVS.FILENAME.TAPE5,DISP=OLD,
             UNIT=AFF=TAPE4
```

The UNIT=AFF option will tell the operating system to load the tape on the same unit as it loaded the referenced tape. In our example, we will need two tape devices, one for TAPEIN and one for TAPE1 through TAPE5, reading them one tape at a time.

This example may be a good use of tape devices, but it is a poor use of processing because we must read the huge source SAS data set five times. Instead, we could use our original DATA step idea, but use TAPE1 for the first of our five output SAS data sets and put the others in temporary disk libraries. Then, we could use the COPY procedure to copy the four temporary SAS data sets to our four other tape libraries, one at a time. Or, we could use five temporary libraries, eliminating the need for all but one tape device. If we do that we will need to alter our UNIT=AFF a bit. Whether we use four or all five temporary libraries, we still use the UNIT=AFF in our JCL to keep the tapes mounted on the same tape device.

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 resources and solutions to those problems presented above are those that I have encountered frequently over my nine years of SAS programming.

REFERENCES

PROC DATASETS:
SAS Procedures Guide, Version 6, Third Edition, Chapter 17
SAS On-Line Documentation, Version 7.1

PROC SUMMARY:
SAS Procedures Guide, Version 6, Third Edition, Chapters 21 and 36
SAS On-Line Documentation, Version 7.1

BLKSIZE
SAS Companion for the MVS Environment, First Edition, Chapter 17
Tuning SAS Applications in the MVS Environment, Michael A. Raithel, Chapter 4

SORT Options:
SAS Companion for the MVS Environment, First Edition, Chapter 17

LIBNAME Statement:
SAS Language Reference, Version 6, First Edition, Chapter 9
SAS Companion for the MVS Environment, First Edition, Chapter 17
SAS On-Line Documentation, Version 7.1

SAS MACRO Variables:
SAS Guide to Macro Processing, Version 6, Second Edition, Chapter 2
SAS Language Reference, Version 6, First Edition, Chapter 20

ACKNOWLEDGMENTS

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: 714-896-6915
Email: casmith@mindspring.com