DON'T DO ANYTHING YOU DON'T HAVE TO: ELEMENTARY STRATEGIES FOR PROCESSING LARGE DATA SETS WITH SAS® SOFTWARE

Yevgenia D. Mackiernan
The Health Data Institute Incorporated,
Baxter Healthcare Corporation

ABSTRACT

This paper discusses the kinds of computer resources large-file processing uses and how to estimate resource requirements to avoid failed jobs. The examples given are specific to batch processing on IBM mainframes running OS, but many of the principles apply to other processing milieux as well. The paper concludes with an example illustrating some of the simplest, most frequently misunderstood, and largest-impact ways to reduce resource use.

INTRODUCTION

The "SAS® System is easy to use--and easy to abuse," so goes the saying. Many of the features of Base SAS software that make it accessible to a great number of people are precisely those features that cause SAS to be a large consumer of computer resources. When SAS software is used inefficiently through lack of care or understanding, the costs can be great. While it is true that there are other languages and systems that use computer resources more efficiently for specific tasks, often the human cost of writing and maintaining such code (not to mention learning the language or system) far outweighs the savings in computer costs. With care and understanding, Base SAS software can be an effective tool for processing large data sets for data analysis.

In general, the more you know about the SAS language and how the SAS System processes data, and the more care you take in designing, testing, and coding your programs, the better you will fare with large data sets. In working with a lot of data, every inefficiency is magnified--sometimes millions of times. Most of the elementary techniques for large data set processing focus on avoiding the unnecessary: Don't make SAS do more processing than it has to. Most beginning SAS programmers use more data steps, create more data sets, read and write more variables, do more sorts, and in general do more processing than the task requires.

LARGE DATA SET TECHNIQUES: A SUMMARY

A great deal has been published over the years in SAS Institute and SUGI literature about using SAS Software with large files. Most of this material falls into one of three categories:

- The importance of planning, testing and using good programming techniques
- Ways to reduce the size of the file being processed through the use of random samples, data summaries, and working subsets
- Ways to avoid excess processing and the relative merits of various processing strategies, particularly with regard to efficient DATA step coding and table look-up techniques.

A checklist of large data set techniques, and a bibliography of SAS Institute and SUGI literature dealing with large data set processing and other suggested readings are presented in the Appendix.

LARGE FILES AND COMPUTER RESOURCES

What is not very often talked about in the literature is, "What do you do when you've done your best to minimize size and maximize efficiency and the remaining file (or job) is still huge?" All computer processing requires space to store data, permanently or temporarily, CPU time for reading, writing and manipulating data, and a certain amount of space (a "region") in the CPU (called memory or core) in which the computer does its work.

After all one's careful planning and testing of efficient code, jobs involving large data sets still often fail in production because they run out of computer resources:

- SPACE to write large permanent data sets
- SPACE to write temporary data sets
- CPU TIME
- MEMORY

Programmers differ in what they consider to be large files. Sometimes large is simply, "the biggest file I've ever processed." Almost always, large means anything that will cost a significant amount of money to process. The kind of large this paper will address is "large enough to make me worry about whether processing it will exceed the upper bounds of my system's resources."

One of the unnecessary things we certainly want to avoid is running the same job multiple times because it lacked sufficient time, space, or memory to complete the first time. In order to work with files this large, one should learn enough about data storage space, CPU time and computer memory requirements to make intelligent estimates of the resources each job will need. The remainder of this paper concentrates primarily on the first resource: space for data storage.

SIZE AND SPACE

The biggest problem with data storage space is that people guess! They do not have a feel for data set size, and they don't know how to translate size into computer terms to request storage space. As a result, users often request what they perceive as "a lot of space" without having any idea how the amount of space requested relates to the amount of space needed for data sets.

To understand space it helps to understand something about what data looks like to the computer and how it is stored.

Data and How They Are Stored

Humans and computers process data at different levels. At the simplest level computerized data consists of bits--binary digits--which humans represent by strings of 0's and 1's. For convenience, clarity or efficiency, humans and computers often process bits in groups of eight: a byte. Bits and bytes are used to encode data in ways which are meaningful to the computer.

Humans think in terms of the larger picture: they cope with bytes by grouping them into fields with conceptual meaning: a 9-byte field for SSN, 6 bytes to contain date of employment, 20 bytes for employee last name. Fields pertaining to a unit of interest--an employee, a hospital admission, an order for automobile parts--are grouped together into logical records or observations. To make efficient use of the computer's capabilities, we usually give it several records at a time, in a BLOCK.

However you see the data, the computer sees it as a succession of bits, which are grouped into bytes, which are grouped into logical records, which are further grouped into blocks.

Figure 1:

The Structure of Data

BITS - represented in binary notation

```
111100001111001111110011111111001
111101101111001011110001111111000
111101001101000111010110110011001000
```

BYTES - the same data grouped into bytes. Under each byte its binary value is represented by two hexadecimal digits.

11110000	11110011	11110011	11111001
F0	F3	F3	F9
11110110	11110010	11110001	11111000
F6	F2	F1	F8
11110100	11010001	11010110	11001000
F4	D1	D6	C8

FIELDS - How the user organizes bytes to represent quantities of interest

```
 9 bytes    Employee ID        Social Security Number
10 bytes    Last Name          All Upper Case
 5 Bytes    Date of Hire       MMDDD (Julian Date)
```

RECORDS - All the fields for an observation

033962184	JOHNSON	86023	...

BLOCKS - A collection of records grouped together for efficiency in processing

Data Storage Media

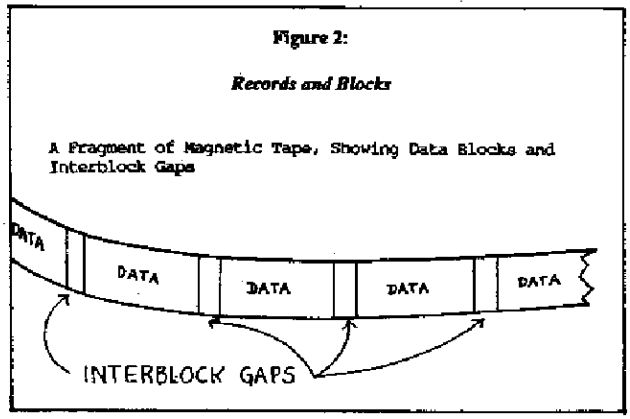
The two most common mainframe data storage media are direct access storage devices (DASD) which are referred to in this paper as disk packs, or volumes, and magnetic tape.

A disk pack or volume is made up of a stack of round metal platters (disks), coated with a magnetic recording material, on which data are recorded in concentric circles called tracks. Tracks placed immediately above each other on successive platters are referred to collectively as a cylinder. There are as many tracks per cylinder as there are platters per disk volume. Each track or cylinder holds a fixed number of bytes, depending on the disk pack model. The maximum storage capacity of a disk volume depends on the number of bytes per track, the number of tracks per platter, and the number of platters per volume. These quantities vary from model to model. Appendix Figure A-1 gives maximum storage capacity for IBM models 3350 and 3380, the most common disk volumes in use on IBM systems today. (The 3380-E has all the characteristics of the 3380, except that it has twice as many cylinders per volume--"double density"--as the original 3380, therefore twice as much maximum capacity.)

Notice that we said "maximum" capacity. What causes capacity to deviate from maximum? Blocking. We said above that records are grouped together into blocks for efficiency in reading and writing operations. Blocking also turns out to be most efficient for storage. When data are written to tape, the block is written as a whole, followed by a space, called an interblock gap, followed by the next block (see Figure 2). A similar process is followed in writing data to disk: the block is written, followed by a gap of a number of bytes, followed by the next block, etc. The difference on disk is that the "gaps" actually contain overhead information that allows the computer to keep track of where all the data is and to access it directly (Direct Access) rather than sequentially. The effect of this system is that only a certain portion of a track is actually available to you for your data, and the rest is used by the operating system for data management.

Figure 2:

Records and Blocks



How much overhead space is needed depends on how many blocks you write, since each block requires overhead, and what device you are writing to, since, as usual, different devices have different requirements. The following table is a simplification of overhead requirements, but it is sufficient for estimates:

Figure 3:

Device Interblock Overhead Requirements

Device:	Overhead or Interblock gap
3350	185 bytes for all but the first block
3380	523 bytes for every block
1600 bpi tape	.6 inches per block
6250 bpi tape	.3 inches per block

Figure A-2 in the Appendix shows the effect of blocking and overheads on device data capacity.

Figuring Storage Requirements

Data set size (in total bytes) and data set storage requirements, while related, are not the same thing. Data set size in bytes is:

SIZE = OBS * BYTES_PER_OBS

Storage is always expressed in terms of the number of storage units required, a storage unit on disk being a track, cylinder or block; storage units for tapes are expressed to the computer as number of tapes. The exact amount of storage needed depends on how the data set is configured: how many bytes per observation, how the observations are blocked, and the size of a storage unit on the device of choice.

Fortunately, we are not usually required to figure the storage units exactly. For our purposes it is sufficient to make an intelligent estimate and allow a margin for error.

To calculate storage for any data set, you need to know:

- How many bytes are in each record?
- What kind of device am I going to use?
- How many records will I group together into a block?
- Disk: How many blocks will fit on a track?
- Tape: How many inches does a block take?

Figure 4 gives formulas and rules used in computing storage quantities. An example follows.

An Example of Storage Calculations. Most of the data you will encounter is recorded as Fixed Blocked (FB) records. Fixed Blocked simply means that each logical record is the same size, and that records are grouped into blocks with a constant number of records per block. This makes size and storage computations quite simple. Even though the SAS System uses a different record format for SAS data sets, the fixed block model can be used to estimate storage. The following example shows a storage calculation for a generic fixed block external data set. The next section describes SAS data sets and gives examples of SAS data set storage calculations.

Suppose you have a file with 80-byte records that you want to store on a 3380 disk pack. You decide to write two blocks per track, because you know (from Appendix Figure A-2) that half track blocking gives you the most data storage space per track on a 3380. This means your maximum block size (MAX_BLKSIZE) is 23476 bytes.

MAX_OBS_BLOCK = MAX_BLKSIZE / LRECL (Use next lower integer)

MAX_OBS_BLOCK = 23476 / 80 = 293.45,

Our answer is 293, because we do not write partial records in a block. A maximum of 293 records will fit in a half-track block on a 3380 device.

[Some people are averse to putting 293 records into a block, because 293 just isn't an appealing number. These people would be apt to use 290 or 280 as a blocking factor. This is fine; with non-SAS data sets, you are in control. It just means you have slightly fewer observations per track. Of course, the nearest "nice" number is 300. However, if you put 300 records per block you get a blocksize of 24000. Which is also fine, IBM computers can handle it, but you will not be able to write 2 blocks per track, only 1, because 24000 is greater than the maximum block size for half track (2-tracks-per-block) blocking. This means almost twice as many tracks of storage would be needed to hold the same data set.]

Let's actually put 293 observations in a block. This gives us a block size of 23440, which is acceptable for half-track blocking. We calculated this according to the formula:

BLKSIZE = LRECL * OBS_PER_BLOCK

= 80 * 293 = 23440

Now all we need to know is how many observations we have—let's say 345292—and we find:

BLOCKS = OBS / OBS_PER_BLOCK (use next higher integer)

and

TRACKS = BLOCKS / BLOCKS_PER_TRACK (Use next higher integer)
= 1179 / 2 = 589.5

The computer allocates tracks in whole numbers, so we know we need 590 tracks. To be safe (Is your arithmetic right? Are you sure there are EXACTLY 345,292 observations?) we usually ask for some extra space, with instructions to release it back to the computer if we don't need it. In your JCL you might code:

SPACE = (TRK,(650,50),RLSE),

asking for at least 650 tracks to start (590 plus a margin) with the possibility of getting more, in increments of 50, if we need it. Anything we don't use will be released at the end of the job.

If your computer center allocates space in cylinders, you get:

CYLINDERS = 590 / 15 = 39.33

Since the operating system does not allocate partial cylinders, ask for a minimum of 40.

Converting from one device to another.

Suppose your data are on 3380 disk and you want to transfer them to tape (at 6250 bpi). How much tape do you need? You could either calculate the number of blocks directly from the formulas (Example A), or, since the blocksize is small, use the conversion factors in Figure 5 (Example B).

Example A

MAX_OBS_BLOCK = MAX_BLKSIZE / LRECL
= 32760 / 80 = 409.5 ==> 409
BLOCKS = OBS / OBS_BLOCK
= 345292 / 409 = 844.23 ==> 845

Example B

TAPE_BLOCKS = 3380_TRACKS * CONV_FACTOR
= 590 * 1.4332 = 845.6 ==> 846

Now calculate footage:

INCHES_BLK = (BLKSIZE/DENSITY) + OVERHEAD
(32720 / 6250) + .3 = 5.5352

TAPE_FEET = TAPE_BLOCKS * INCHES_BLK / 12
= 846 * 5.5352 / 12 = 390.23

If your tape blocks are close to 32760 you can simply use 5.5416 and 21.075 as INCHES_BLK for 6250 bpi and 1600 bpi tapes respectively.

Figure 5:

Converting Storage Requirements Between Device Types

For small record lengths and maximum blocking (half-track on 3380), the following factors may be used to convert between 3380 tracks, 3350 tracks, and tape blocks. For larger logical records, the estimates become less precise.

	TO: 3380 TRACKS	3350 TRACKS	TAPE BLOCKS
FROM:			
3380 Tracks	——	2.4622	1.7180
3350 Tracks	.4061	——	1.4332
Tape Blocks	.5821	.6977	——

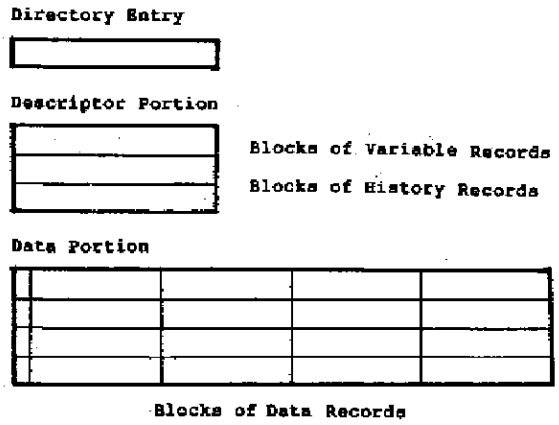
CALCULATING STORAGE FOR SAS DATA SETS

The Structure of SAS Data Sets

On IBM/OS systems, a SAS Data Set consists of three parts: A directory entry, a descriptor portion, and a data portion. Figure 6 shows a schematic representation of the three parts of a SAS data set. Formulas for computing the amount of space required by each part are given in the Basics manual on pp 36-38. These formulas can seem overwhelming: The novice often gives up and goes back to guessing. Do not despair! For our purposes it is not necessary to know exactly how many bytes and tracks are needed. Estimate, and add a margin for error.

Figure 6

Schematic Representation of a SAS Data Set



In working with large data sets, the space required for the directory entry is negligible. The space required for the descriptor portion is usually, but not always, insignificant compared to the space required for the data portion itself. Therefore in calculating storage requirements for SAS Data Sets, we can simply calculate the space required for the data portion and add a small margin for the rest.

Storage for SAS Data Sets

Calculating storage for SAS Data Sets can be very simple. Every time you create a SAS Data Set on disk, your SAS Log contains a note:

NOTE:THE DATA SET OUT1.MASTER HAS 1000 OBSERVATIONS AND 324 VARIABLES.18 OBS/TRK.

This tells you how many data observations fit on one track of storage space. To calculate storage for the entire data set on the same type of device, you only need to know the total number of observations you expect in the final data set. Suppose we expect 400,000 observations.

TRACKS = OBS / OBS_TRACK.
≈ 400,000 / 18 = 22,222.2 ==> 22,223

We will want to add small a margin for directory and descriptor records. Let's ask for 22500.

Now we're in trouble! Appendix Figure A-1 tells us that there are only 13275 tracks on an entire 3380 disk pack. One of the limitations of SAS Data Sets on disk is that they cannot span disk volumes, that is, the entire data set must fit on one disk volume. If we have a 3380-E, which has twice as many tracks as a 3380, we could write OUT1.MASTER to disk.

Still, 22500 tracks is 84.8% of a 3380-E. You need to know: Is there such a volume available to you that is 84% empty? What will this storage cost you? At charge-back installations, disk storage is usually $.01-$.04 per track per day unless whole pack rentals are arranged (usually several thousand dollars per month). Even at a penny per day, OUT1.MASTER is costing at least $225 a day to store. Many computer centers monitor storage of data sets as small as 200-500 tracks in order to keep public storage space free for all users. Non-chargeback systems are usually more aggressive than chargeback systems in this regard. You may find your data center reluctant to give you this much space unless you lease a disk pack long term.

If you don't need to use the direct access (POINT=) feature of SAS Data Sets (or other disk-only features) on a daily basis, you should store OUT1.MASTER on tape. While tape storage can be less flexible and less convenient than disk storage, it is VERY CHEAP in comparison to disk storage.

How do we figure how many tapes we need? One handy statistic, is that at maximum blocking a 6250 bpi tape contains about 170 megabytes of data (see Appendix Figure A-1). If we knew the size of the full OUT1.MASTER in megabytes, we could make a quick estimate. We know that the maximum effective track capacity of a 3380 is 46952 bytes (Appendix Figure A-1 again). So the most our 22223 tracks can contain is 22223 * 46952 = 1043.4 megabytes. At 170 megabytes per tape, that's 6.13 tapes. You would allocate minimum of 7 tapes (2400 foot) minimum for this job. You'd be safer (especially if your installation is not in the habit of mounting scratch tapes) to allocate an extra tape for good measure. Among other things, estimates are exactly that--estimates--and some are better than others. Also, the exact usable length of a 2400 foot tape is not always 2400 feet!

We could be more precise by using the formulas in Figure 4. To do this we would like to know the LRECL and BLKSIZE that SAS will use. While SAS Data Sets are technically RECFM = U (record format = unformatted), the SAS System uses a fixed LRECL and BLKSIZE for the data portion of the data set. The data portion LRECL is the sum of the lengths of all the variables, plus 4 bytes. The BLKSIZE used depends on the value of the BLKSIZE= option set at your installation. You can find both the LRECL and the BLKSIZE used for OUT1.MASTER from a PROC CONTENTS on the data set.

Special Storage Considerations for SAS Data Sets

Remember that when you allocate storage space for a new SAS file you are actually allocating space for a SAS Data Library which can contain more than one SAS Data Set. The SAS Data Library needs to be big enough to contain all the data sets that will be stored there.

If you are sorting a SAS data set without an OUT= option, the SAS system stores the sorted records temporarily in the same data library as the original file. Then, when the sort completes properly, the old version is erased and replaced by the sorted version. If you were to try to sort OUT1.MASTER:

```
PROC SORT DATA = OUT1.MASTER;
  BY PERSONID;
```

You would need twice 22500 tracks in the data set referenced by OUT1. If you use OUT= with your sort you still need twice as much space, but not all in one data library:

```
PROC SORT DATA = OUT1.MASTER OUT = OUT2.SORTMAST;
  BY PERSONID;
```

Now you need 22500 tracks for OUT1.MASTER, and 22500 tracks for OUT2.SORTMAST! Note that you must use OUT= when sorting tape data.

The same thing happens when you replace a SAS data set with itself (a practice this author does not advise).

```
DATA OUT1.MASTER;
  SET OUT1.MASTER;
```

The OUT1 library temporarily needs space for two copies of the member MASTER. In our example, you cannot get enough space due to the single-volume restriction.

Descriptor Records

Occasionally, the Descriptor portion of a SAS data set can become extremely large. Among the material that is stored in the descriptor portion is information about each variable in the data set, and data set history, that is, source code for the current data set and every preceeding data set back as many generations as the GEN= option allows. (Use PROC OPTIONS to find out your system default for the GEN= option.) If your data set is a large one and comes from a long line of large merged and constantly updated data sets, history information alone can consume several hundred tracks. SAS uses 90 bytes of storage per line of source code, blocked at 1024 or less. This allows only about 300 lines of code per track.

If you suspect you have generation troubles, look at your test data set. Does it occupy many more tracks on disk than your calculations suggest? Try rerunning the test with OPTIONS GEN = 0. Does the space occupied change? Now you know how much space to allocate for the non-data portion of your data set, or you may chose to economize on storage by running the production run with GEN = 0;

Unknown Quantities

Knowing how many observations you will have is sometimes hard to predict. You should at least know how many observations are in every infile or data set you will read. If you don't know, you can get a good estimate for INFILEs from the LRECL and BLOCK COUNT on a PROC TAPELABEL (if tape). If your INFILE is on disk, there is usually a utility which will tell you LRECL, BLKSIZE and tracks. This gives you enough information to estimate.

Even for a SAS data set on tape, PROC CONTENTS will tell you the LRECL and BLKSIZE, but not the number of observations. Using the block count from a PROC TAPELABEL will yield an overestimate of the number of observations, because that block count includes all the directory and descriptor records.

What about new large data sets you are creating? How many observations will they have? If you intend to keep all your observations then you have your answer. If you intend to subset or expand the data or to merge, you will have to extrapolate from test runs. You will be best served if you have been using a random-sample test file, as such files are most likely to produce accurate extrapolations. Beware of extrapolating from an OBS = 100 run. For one thing, bad data always seem, perversely, to wind up at the top or the bottom of the file. Secondly, subsetting observations usually depends on the values of certain variables. If you use a random sample you avoid the possibility that all observations with a given value of a subsetting variable are grouped together in the file.

Developing a Feeling for Size and Space

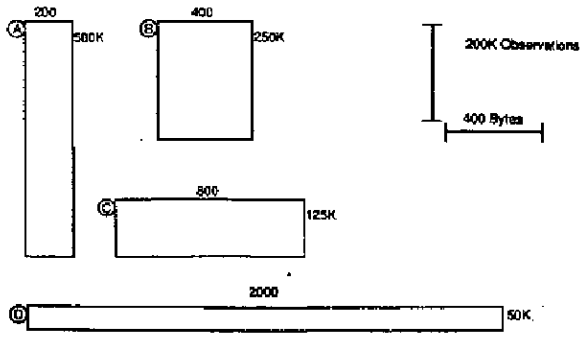
When asked to define "large", people most often answer only in terms of the number of observations--50,000 obs, a million obs--forgetting that files have two dimensions, observations and variables. How big is each observation, that is, how many bytes of data does it contain? Figure 7 illustrates several different file configurations, each of which contains 100 million bytes (100 Megabytes) of data.

Figure 7

Observations, LRECL and Variables in Four 100-Megabyte Data Sets

DATA	OBS	LRECL	VARS*
A	500K	200	25
B	250K	400	50
C	125K	800	100
D	50K	2000	250

* Number of 8-byte numeric SAS variables which this record length could contain. For data records, SAS uses an LRECL equal to the sum of all the variable lengths, plus 4 bytes.



These data sets are all the same "size" in terms of total bytes of data, although they have slightly different storage requirements (see Figure 8).

When we talk about the number of SAS variables in an observation, we are only giving a partial answer, because we do not know how long each variable is. Numeric variables can be from 2 to 8 bytes in length, and character variables from 1 to 200 bytes. When talking

about the size of a SAS observation, we need to know its length in bytes, or LRECL (logical record length), which is simply the sum of the storage lengths of each variable, plus 4 bytes of overhead. PROC CONTENTS gives you this information. The convention in SAS manuals is to talk about variables as though they were all 8-byte numerics: 100 variables often means an LRECL of 804.

Space for Temporary (Work) Data Sets

Another potential space problem with large data sets is running out of space for temporary data sets created during your job.

When you run an OS batch job in SAS, the SAS System allocates a temporary SAS Data Library with the DDNAME WORK. You can see this allocation in the expansion of the SAS PROC at the top of your IBM job log. At our installation it looks like:

```
//WORK DD UNIT=SYSDA,SPACE=(6160,(750,250))
```

This statement allocates up to 27.7 megabytes (about 590 tracks on a 3380) for the SAS Data Library WORK.

The WORK Library is used by the SAS System to contain temporary utility files and temporary second copies of SAS data sets created while a sort is in progress or a data set is being replaced. These files remain in the library only until the end of the relevant DATA or PROC step. However, when you create a SAS data set using a one-level name, as in:

```
DATA STEP1;
  SET ...
```

the SAS System places a SAS data set called STEP1 into the WORK library, where it remains until your job ends. The WORK library needs to be big enough to hold simultaneously all the temporary data sets you create in your job. If you are reading large files into temporary data sets, this can be a lot of space.

Now that you know how to calculate the space required for one data set, you can calculate the space required for each work data set as well. (In practice, you only need to worry about the large ones.) Just as you can request space of the system for your permanent SAS data sets, you can request space for any temporary data set and for the WORK data library.

There are several strategies for managing space for temporary data sets.

Change the amount of space allocated to the WORK library with a //WORK DD statement placed immediately after the EXEC SAS card.

Create your own "work" library, by using OPTIONS USER=MYLIB, where MYLIB is the DDNAME of a large SAS data library. The option USER= causes the SAS System to change the default library into which it writes data sets with one-level names. DATA STEP1; becomes the in the JCL

Use separate DDNAMEs for large temporary data sets and allocate space for them in the JCL. When you have large temporary data sets you may want to consider splitting up the job into smaller steps. Or, you can use the DISP parameter (NEW,DELETE,CATLG) to delete disk data sets automatically if the job terminates normally and to keep them if the job abends. That way you may also avoid having to rerun your job from the beginning.

Write very large temporary data sets to tape. Some installations have scratch tapes which can be mounted for this purpose; at others you have to use your own tapes.

Consolidate your data steps and use KEEP= and DROP= as Data Set options whenever possible. This keeps temporary data sets small, and most novices use too many data steps anyway

Use PROC DATASETS to delete temporary data sets from the WORK library when they have served their purpose and the job could continue from this point (even if it failed) without them.

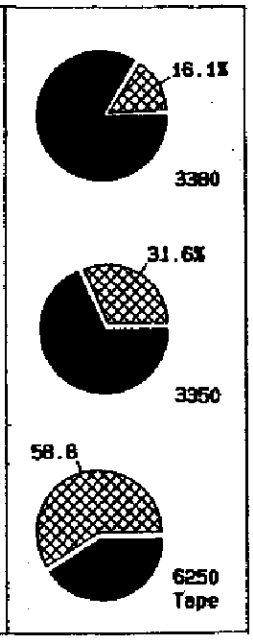
Figure 8

Storage Required on Disk and Tape for Four 100-Megabyte Data Sets

DATA SET	OBS	LRECL	SAS VARS	3380* OBS/TRK	3380* TRACKS	3350* OBS/TRK	3350* TRACKS	TAPE** OBS/BLOCK	TAPE** BLOCKS
A	500K	200	25	234	2137	95	5264	163	3068
B	250K	400	50	116	2156	47	5230	81	3087
C	125K	800	100	58	2156	23	5435	40	3125
D	50K	2000	250	22	2273	9	5555	16	3125

* Track requirements are shown using half-track blocking on the 3380, and full track blocking on the 3350 and tape.

** 3068 tape blocks of 32600 bytes each takes approximately 1411 feet of tape at 6250 bpi and 5362 feet at 1600 bpi. Tape is most commonly encountered in 2400-foot reels, although 1800, 1200 and 600 foot versions are also used.



CONCLUSIONS

In any profession, a worker must expect to learn the tools of the trade in order to do a good job. As a SAS programmer you should study the workings of the SAS language and the environment in which it operates. You need to know what tools are available to you and how to use them; you need to know your own skill level – what you can safely do by yourself and where you need help.

Mistakes with large data sets are costly, wasting both computer resources, and your own time and that of others. While many issues involving the processing of large data sets, such as designing an appropriate and efficient use of the data, are advanced skills, there is much the novice can do to improve the efficiency of large-data programs. Every novice can learn to write consolidated efficient data steps that read and write as little data as seldom as possible. The novice should develop an understanding of data set size and be sure that SAS jobs are given adequate resources to run correctly the first time!

ACKNOWLEDGMENTS

The author thanks Ron Liebman of the Health Data Institute for several helpful suggestions during the preparation of this paper. This paper is published in the Proceedings of the Fourteenth SAS Users Group Conference, Cary, NC: SAS Institute, Inc. SAS is a registered trademark of SAS Institute Inc., Cary, NC, USA. The author may be contacted at:

Yevgenia D. Mackiernan
Director, Quality Assurance
The Health Data Institute
Baxter Healthcare Corporation
20 Maguire Road
Lexington, MA 02173

REFERENCES

Brown, Gary deWard. System 370 Job Control Language. New York: John Wiley & Sons, 1977.

International Business Machines Corporation. OS/VS2 MVS Data Management Services Guide, Release 3.8, Second Edition. IBM Publication GC26-3875-1, 1980.

Lowe, Doug. MVS JCL. Fresno, CA: Mike Murach & Associates, 1987.

SAS Institute, Inc. SAS Applications Guide, 1987 Edition. Cary, NC: SAS Institute Inc, 1987.

SAS Institute, Inc. SAS User's Guide: Basics, Version 5 Edition. Cary, NC: SAS Institute Inc, 1985.

APPENDIX

Suggested General Readings

SAS User's Guide: Basics (Version 5 Edition):

Chapter 2 - Introduction to the SAS Language, pp 11-21
Chapter 3 - Introduction to the DATA Step, pp 25-38
Chapter 4 - The LENGTH Statement, pp 152-157
Chapter 17 - SAS Files, pp 553-573

SAS Applications Guide (1987 Edition):

Chapter 10 - Processing Large Data Sets with SAS Software, pp 223-238

A Bibliography of SUGI Readings on Large Data Set Processing

The following papers from past SUGI conferences, organized by year, may be found in the Proceedings volumes (Proceedings of the __th SAS User's Group International Conference, Cary, NC: SAS Institute Inc.). At this writing all of these proceedings are still available from the publications department of the SAS Institute Inc. The papers cover a variety of topics of interest to the processor of large data sets.

1988 - SUGI 13

Caron, A., "Elements of Good Programming Style" 309.
Davis, G.W. and S.S. Sweetland, "Efficiency and Performance Considerations for the SAS System under PC DOS" 779.
Fox, N., and C. Ray, "Extraction from Enormous Sequential Health Care Claims Data Files Using Multiple Format Tables" 561.
Henderson, D.J. and M. Rabb, "The SAS Supervisor" 1161. (This paper is in every SUGI proceedings since 1983.)
Horwitz, L. and E.T. Thompson, "Efficient Programming with the SAS System for Personal Computers: Save Time, Save Space" 625.
Howard, N., "Cutting Costs in the DATA Step: Efficiency Techniques for Improving I/O and Processing Time" 1178.
Langston, R. "Numeric Precision Considerations in SAS Software", 635.
Lutomski, L.S. "Efficient Merge Operations for Historical Data" 595.
Lutomski, L.S. and H.A. Mullally, "The Comparative Efficiency of Table Look-Up Methods" 648.
Ma, J.M., "Processing Large Files: The SUMMARY Procedure and Other Efficiency Tips" 1279.
Mahan, R.L., "Sort Efficiency Using the SAS System Under VM/CMS" 122.
Ray, C., "Implementation of a Hashing Routine in SAS Software" 1184.
Reel, J., "A Flow Chart for Users to Solve SAS Programming Problems" 267.
Rubin, D., "Sorting Massive Data Sets" 1252.
Squillace, D.J., "Producing More Efficient Applications" 82.
Svirsky, V., "Efficient Use of a SAS Data Library as a Pseudo-Relational Data Base" 107.
Thornton, R., and M. Rabb, "Advanced SET and MERGE Processing" 1168.
Virgile, R., "Low-Maintenance Programming Techniques" 1242.
Young, J.P. "Strategies for Shringking PC Data Sets and Finding More Free Bytes" 613.

1987 - SUGI 12

Fischell, T.R. and E.G. Hamilton, "Processing Large Data Sets into Customized Tables" 978.
Hamer, R.M., "Using the SAS System with Relational and Hierarchical Data" 86.
Howard, N., L.W. Pickle, and J.B. Pearson, Jr., "Effective Methods of Testing Using the SAS System" 400.
Kolman, J. "The Design and Programming of Large Systems using SAS Software" 587.
Kretzman, P. "Lean and Mean: A Guide to Self-Discipline in Using the SAS System" 144.
Ma, J.M., "How to Use SAS Software Effectively on Large Files" 94.
Marsh, P., "Optimizing IBM-PC Systems for SAS" p 775.
Norton, A. "Getting the Most from PROC TABULATE" 118.
Ray, C., "A Comparison of Table Lookup Techniques" 40.
Squillace, D. "Tuning the SAS System under CMS" 217.

1986 - SUGI 11

Croy, C.D., "What SAS Software Does and Doesn't Do: Things Your Mother Never Told You." 895.

Konowal, L. and S. Clark, "Efficient Use of PROC SUMMARY", 664.
Ray, C., and H. Levine, "Efficient Use of Table Lookup Procedures" 718.

1985 - SUGI 10

Dineley, V.M., et al., "Writing Efficient SAS Data Steps" 684.
Lafler, K.P. and V.E. Kirchman, "Optimization Techniques for SAS Applications" 530.
Ma, J.M. and T.R. Fischell, "Descriptive Statistics: Using Indicator Variables" 1026.
Mopsik, J. and A. Asher, "Enhancing SAS Software Skills Through a Better Understanding of the SAS Log" 1206.
Rosenberg, M.J. and C.A. Bogardus, "Optimizing Storage of SAS Data Sets" 334.

1984 - SUGI 9

Asher, A.M., and M.C. Eley, "Improving Programmer Productivity Through Coding Conventions" 682.
Howard, N., and L.W. Pickle, "Efficient Data Retrieval: Direct Access Using the Point Option" 294.
Ma, J.M., "PROC SUMMARY as the Basis for Efficient Analysis of Large Files" 309.
Merlin, R.Z., "Design Concepts for SAS Applications", 283.
Ramsay, A., "Keyed Access to SAS Data Sets" 322.
Sharlin, J. "Data Reduction and Summarization" 912.

1983 - SUGI 8

McGregor, S.L. and M.Nelson, "Some Guidelines for Documenting SAS Source Code" 119.
Muller, K.E., J.C. Smith, and J.S. Bass, "Managing 'Not Small' Datasets in a Research Environment" 371.
Mopsik, J., "Efficient Techniques for Large Data Files" 950.

1982 - SUGI 7

Henderson, D., "Conditional Execution of Data and Proc Steps" 787.
Henderson, D., "Selecting Subsets of Data" 799.
Henderson, D., "Table Lookup Techniques" 792.
Rikard, P.L., "Unconventional Merges" 838.
Sharlin, J., "Improving the Data Management Expertise of Experienced SAS Users" 100.

1981 - SUGI 6

Muller, K.E. and D.H. Christiansen, "Rules We Followed and Wish We Had Followed in Managing Datasets, Programs and Printouts" 401.

Figure 4:

Formulas for Computing Storage Requirements

DEFINITIONS:

BLKSIZE	=	Block size: the number of bytes actually contained in a block of records
BLOCKS	=	The number of blocks present (an integer)
CYLINDERS	=	The number of cylinders of storage needed. Calculated from tracks. See Appendix Figure A-1.
DENSITY	=	The density at which data are written to tape. Most commonly 6250 bpi (bytes per inch), although 1600 bpi is also used.
LRECL	=	Logical record length. The number of bytes in a record, usually the sum of the variable lengths or that sum plus 4 bytes. Available for SAS data sets from PROC CONTENTS.
MAX_BLKSIZE	=	Maximum block size, for a given device and blocking scheme (see Figure 3).
MAX_OBS_BLOCK	=	Maximum number of whole observations that fit in a block of choice. Calculated.
OBS	=	Number of observations in the data set.
OBS_PER_BLOCK	=	Actual number of observations in a block. May differ from MAX_OBS_BLOCK because some programmers prefer to use a slightly smaller round number than the maximum possible.
OBS_PER_TRACK	=	Number of whole observations which fit on a track (calculated, or from PROC CONTENTS or the SAS Log)
TRACKS	=	Number of tracks needed (an integer)
TRACKS_PER_CYL	=	Number of tracks contained in a cylinder. Device dependent, see Appendix Figure A-1.
TAPE_BLOCKS	=	Number of tape data blocks being written.
TAPE_FEET	=	Number of feet of tape required for a data set at a given density.
TAPE_INCH_BLK	=	Number of inches of tape required per tape block at a given density (calculated).

FORMULAS: (See examples in the text.)

BLKSIZE	=	LRECL * OBS_PER_BLOCK
BLOCKS	=	OBS / OBS_PER_BLOCK (Use next higher integer)
CYLINDERS	=	TRACKS / TRACKS_PER_CYLINDER (Use next higher integer)
MAX_OBS_BLOCK	=	MAX_BLKSIZE / LRECL (Use next lower integer)
TRACKS	=	BLOCKS / BLOCKS_PER_TRACK (Use next higher integer)
TRACKS	=	OBS / OBS_PER_TRACK (Use next higher integer)
TAPE_BLOCKS	=	3380_TRACKS * CONV_FACTOR (Use next higher integer)
TAPE_FEET	=	BLOCKS * TAPE_INCH_BLK / 12
TAPE_INCH_BLK	=	(BLKSIZE/DENSITY) * OVERHEAD

RULES:

An integral number of observations are written to a block.
An integral number of blocks are written per track.
A block may not be larger than a track.
A SAS data set on disk must be wholly contained on one disk volume.

Figure A-1

Maximum Device Capacity

DISK

DEVICE TYPE	3350	3380	3380 EFFECTIVE*
Maximum Bytes per Track	19,069	47,476*	46,952
Tracks per Cylinder	30	15	15
Cylinders per Volume	555	885	885
Tracks per Volume	16,650	13,275	13,275
Maximum Bytes per Volume	317.5M	630.2M	623.2M
Device Overhead (bytes per block)	185	523	

* The maximum blocksize supported by IBM is 32760. A better use of 3380 space is to write two blocks, of up to 23476 bytes, per track. This gives a maximum track capacity of 46952 bytes.

** This is a simplification, but good enough for estimates. Use 185 for bytes for all but the first block on a 3350, 523 for all blocks on a 3380.

TAPE

TAPE RECORDING DENSITY	6250 BPI	1600 BPI
Maximum Bytes per Block	32760	32760
Inches per 32,760 Block	5.2416	20.475
Interblock Gap (Inches)	.3	.6
Blocks on 2,400 Feet of Tape	5197	1366
Bytes on 2,400 Feet of Tape	170.3M	44.8M

Figure A-2

Effect of Blocking on Device Data Capacity

3880 DISK VOLUME

BLOCK TYPE	BLKSIZE	BYTES/TRACK	BLOCKS/TRACK	BYTES/VOLUME
Full Track*	32760	32760	1	434.9M
Half Track	23476	46952	2	623.3M
Third Track	15476	46428	3	616.3M
Fourth Track	11476	45904	4	609.4M
4096	4096	40960	10	523.7M
1024	1024	30720	30	407.8M

*Largest blocksize supported by IBM/OS is 32760 bytes.

3350 DISK VOLUME

BLOCK TYPE	BLKSIZE	BYTES/TRACK	BLOCKS/TRACK	BYTES/VOLUME
Full Track	19069	19069	1	317.5M
Half Track	9444	18888	2	314.5M
Third Track	6234	18702	3	311.4M
Fourth Track	4629	18516	4	308.3M
4096	4096	16384	4	272.8M
1024	1024	15360	15	255.7M

2400 FOOT TAPE VOLUME

BLOCK SIZE	6250 BPI BLOCKS/ VOLUME	6250 BPI BYTES/ VOLUME	1600 BPI BLOCKS/ VOLUME	1600 BPI BYTES/ VOLUME
Maximum (32760)	5197	170.3M	1366	44.8M
10000	15157	151.6M	4204	42.0M
4096	30145	123.5M	9113	37.3M
1024	62090	63.6M	23225	27.6M