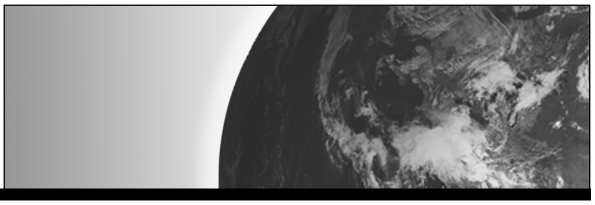
Chapter 1



Resource Tips

Some ways to save disk space

Disk space equates to cost. Reducing space means the need for less disk hardware and therefore less cost.

1. Use PROC DATASETS, the DIR window, or the Libraries window to delete temporary and permanent data sets after use.
2. Use the KEEP= or DROP= data set options to limit the data set to only the variables required. You can also use the KEEP or DROP statements to do this.
3. Use the WHERE= data set option or WHERE statement to limit the number of observations processed by procedures or DATA steps. A WHERE statement can be used to replace an IF statement in a DATA step, and can be more efficient.
4. Use _NULL_ as the data set name in the DATA statement when you don't need to create a data set—for example, when creating a report.
5. Use remote library services to enable you to keep only one copy of data on your network.
6. Use data set compression. Data set compression can be done in one of several ways. Using either a data set option, a LIBNAME option, or a system option, you can set the compression to use. COMPRESS=YES is equivalent to COMPRESS=CHAR, and is good for compressing data that has mainly character values. COMPRESS=BINARY is good for compressing data that has mainly numerics. COMPRESS=NO disables compression.
7. Use views, rather than temporary data sets, but remember that this will increase CPU time.
8. Use pipes to compress data (for operating systems that support pipes). These enable compressed data to be read and written in real time. Please see your operating system companion for more information.
9. Use SQL to merge, summarize, sort, and so on, rather than using a combination of procedures and DATA steps with temporary data sets. Using one SQL statement can avoid saving temporary data sets, depending on the data that you are using. Keep in mind that SQL is often not as efficient as DATA step and procedure code.
10. Keep temporary files on tape, cartridge, CD, DVD, or other high-capacity media.

11. Use SQL pass thru for relational database processing to allow SQL to use temporary file space of the server SQL system (which is often a larger complex). This avoids using space on the machine that you are running.
12. Produce a format and store coded values in your data set. These values can be decoded using the format in a DATA step or a procedure.
13. Put your data into something approaching third-normal form, although this can affect system performance. Third-normal form involves splitting data (where appropriate) into more tables where tables with data that have a 1:1 relationship are placed in individual tables. For instance, with address data, you could have a table of zip codes and city names, which would mean the main address data could have zip codes but not city names. Then, when you want to know a city name, you could just take the zip code and look it up in the zip/city file.
14. Store your data in the order in which it is usually required. This avoids the need to re-sort data, thereby saving utility work space. Indexes can be added to avoid re-sorting too. Generating and maintaining indexes can take resources. Also, the index itself will take some space and use CPU time.
15. If you have a large SAS program consisting of many steps, then when reading a file into a data set, you should delete any observation that you don't need as soon as you determine you don't need it. This reduces the size of the data set when it is subsequently used in the rest of the program.
16. Use the LENGTH statement to limit the bytes used to represent a number or character to what is required for the desired precision. You must be careful when doing this for numerics, because the precision can be affected and CPU time will increase slightly. It is safest to do so only for characters and integer numerics, unless you are sure of what you are doing. Consider taking the SAS Programming III course.
17. An ideal technique for reducing the space required to store SAS dates is to reduce the length of the variable to 4. You require only 4 bytes of storage, not 8, to store any date.
18. Minimize the data that you keep in your permanent SAS libraries. Always ask yourself several questions:
    - Do I need this data? (What value is the information that it represents?)
    - Can data within the data set be derived from other information that I have? (For example, don't keep month, year, and day if you have a date.)
    - Is there another copy that I can refer to (for example, data held in DB2)?

- Is the cost of reproducing the data high (i.e., can I re-run my SAS program to reproduce the data)?
- Am I likely to need to refer to this data before it is outdated? (Daily data may be useful only for a day.)
- How much is it costing to keep this data (disk charges, etc.)?
- Can I tell the system to delete my data when it is no longer of use (for example, automatically delete data after 30 days)?
- Can I summarize historical data and delete details if I never again need them?

19. Delete any unused indexes. Make sure that the indexes are not being used by anyone before deleting.
20. Store program code centrally, rather than distributing it to users. Maintaining one copy will save space and make maintenance easier.
21. If the length of numeric ID variables is more than 8 digits (**Note:** default numeric length is 8), save it as a numeric variable, such as account number, Social Security number, employee, ID or student ID (all numbers). This also applies for 8 or fewer digits. For example, a number using 3 bytes of storage (in Windows) can represent a 4-digit number up to 8,192, and a number using 6 bytes of storage can represent a 12-digit number up to 137,438,953,472. For example,
    - Don't use: Length ID $ 16 ;
    - Instead use: Length ID 8 ; which will save half the space!
22. Define an index to avoid sorting. (Sorting often takes very large amounts of space.) Of course, indexes use space, but often the space used can be far less than that required for sorting. Indexes are not as efficient as sorted data, however.
23. Delete records not needed when they are read.
24. Don't use audit trails unless you need them.
25. Keep data in permanent libraries only if it will be needed later.
26. Use character variables to store numbers. For instance, if a number will only ever be 0 or 1, then it can be stored in a character of length 1, but a number (under Windows) must be at least 3 bytes long.
27. Use the FTP filename access method to access data on other machines, rather than making a local copy.

28. Use the LENGTH function to shorten characters and numerics when possible. You can write a program to analyze a data set and set lengths appropriately based on data values.

29. Use the SASFILE statement to load data sets into memory, which saves some space that you would otherwise need for them on disk. This also has the benefit of giving faster access to the data set in memory.

How to save space in SAS catalogs

SAS catalogs are not automatically compressed. As you save catalog entries, unused space accumulates. In some cases, less than half the space used by a catalog is actually needed.

To compress and reuse the unused space in a catalog, use REPAIR in PROC DATASETS.

This will compress sasuser.profile:

```
Proc datasets library=sasuser ;
  repair profile / mt=cat ;
quit ;
```

9 ways to minimize input/output

Input/output (I/O) to disk is the factor that usually slows down SAS programs. Reducing I/O will speed up execution and often reduce costs.

Generally, SAS is I/O intensive, rather than CPU intensive. As the great performance and tuning guru Ken Williams says, “The best I/O is the one you didn’t do.” Thus, a saving in I/O will improve the performance of your SAS program.

Here are 9 ways to minimize I/O:

1. Use LENGTH statements to minimize variable lengths, where possible.
2. Use CLASS statement(s), where possible, rather than BY statements, which might require a SORT.
3. Use DROP and KEEP statements to minimize observation length.
4. Use SORT only when necessary.
5. Create multiple data sets in one DATA step when possible.
6. Use WHERE statements with procedures to avoid subsetting in a DATA step followed by a procedure.
7. Use the _NULL_ DATA step when you don’t need to create a data set. For example, you might want to create an external file, produce a report, or just do some calculations.
8. Compress some large SAS data sets, but beware that compression can use more space in some cases, which might actually increase I/O.
9. Develop and test programs on a small subset of the data.

Note: This list is not comprehensive. It merely attempts to provide a few ideas for investigation. Not all of the points will always reduce I/O time.

Implementing application data security

The aim is to let permitted users access data via your application, but to make it very difficult for anyone to access data without your application. The application should detect who the user is and should provide appropriate data access for them.

Data set passwords

You can put passwords on SAS data sets. This prevents accessing them without specifying the password. This also prevents users from accessing SAS data sets in their own batch jobs (unless they know the password). Passwords can be coded into source code so that your program "knows" the password. Using PROC PWENCODE is useful in this case, because it will generate a string, which cannot be recognized as the password, in place of the password.

Operating system security

On z/OS, you may have RACF or ACF2 to secure your data sets. On Windows and UNIX, you can protect directories from unauthorized users. On standalone PCs, you can often specify a startup password.

SAS®9

In SAS®9, the Business Intelligence Architecture enables much more security. By using the SAS Metadata Server, you can define users and groups of users along with information about the resources that they can access and what they can do with them.

Data encryption

An encryption key (or algorithm) can be used to encode numbers or text and can be kept in a secure data set. It can be read in when compiling the application or can even be built by an algorithm in the code. Different encryption keys can be used for different groupings of information to add another level of security. This means that a hacker would need many encryption keys to access all of the data.

There is also the ENCRYPT data set option, which makes SAS encrypt your data sets.

Other points

- You should exit SAS before the application ends.
- Close secured files after they have been used. Free FILENAME and LIBNAME allocations when you are finished with them.
- For z/OS, specify the NOSTAX system option so that the attention key will end the SAS session.

Useful options for tuning

When tuning your SAS program to make it run more efficiently, it is useful to turn on various information options available in SAS. Remember to turn them off when you finish the tuning and run your program, because many options increase the overhead (CPU time, elapsed time, I/Os) of your program.

```
options oplist stats fullstats echoauto source source2 memrpt mprint
stimer ;
```

Option	Description
oplist	Shows settings of SAS system options in SAS log
echoauto	Shows autoexec file in log
stats	Writes performance statistics to log
fullstats	Writes performance statistics in expanded form
stimer	Maintains and prints timing statistics (Don't use this with views.)
memrpt	Shows memory report
source	Shows source code in log
source2	Shows included source code in log
mprint	Shows statements generated by macro facility

Also consider using ARM macros or the Rtrace facility for other approaches to tuning.

Saving resources when the log is long

When writing a lot of information to the log in interactive SAS, you can be slowed down as SAS scrolls the log to display each line as it is written. This is the default behavior, which works well when you don't write much to the log. This tip tells you how to save time and resources by altering the AUTOSCROLL setting.

Activating the Log window (by selecting Log from the View menu or by clicking the Log window) and setting AUTOSCROLL to 0 tells SAS not to bother scrolling the Log window until the DATA step is finished. AUTOSCROLL can be set by using the pull-down menus to choose EDIT then OPTIONS then AUTOSCROLL.

The following example takes 31.25 seconds to run (on my test machine) with AUTOSCROLL=1. Then, running the same code with AUTOSCROLL=0 takes only 0.29 seconds. I ran these tests several times to ensure there was no effect due to caching of data or anything due to the order of the code being run.

```
dm 'log; clear;autoscroll 1' ;
data _null_ ;
  set sashelp.prdsale ;
  do i=1 to 50 ;
    put year= month= actual= ;
  end ;
run ;
dm 'log; clear;autoscroll 0' ;
data _null_ ;
  set sashelp.prdsale ;
  do i=1 to 50 ;
    put year= month= actual= ;
  end ;
run ;
```

```
568    dm 'log; clear;autoscroll 1' ;
569    data _null_ ;
570      set sashelp.prdsale ;
571      do i=1 to 50 ;
572        put year= month= actual= ;
573      end ;
574    run ;

Lines deleted

NOTE: There were 1440 observations read from the data set
  SASHELP.PRDSALE.
NOTE: DATA statement used (Total process time):
      real time           7.81 seconds
      cpu time            7.39 seconds

575    dm 'log; clear;autoscroll 0' ;
576    data _null_ ;
577      set sashelp.prdsale ;
578      do i=1 to 50 ;
579        put year= month= actual= ;
580      end ;
581    run ;

Lines deleted

NOTE: There were 1440 observations read from the data set
  SASHELP.PRDSALE.
NOTE: DATA statement used (Total process time):
      real time           0.29 seconds
      cpu time            0.29 seconds
```

Several ways to tune a SORT

Here is a brief list of things you can look at if you want to make your PROC SORT go faster, use less space, or use less CPU time. Always test the methods and combinations to see what works best for you.

- SAS®9 can use a product called SyncSort to perform sorts, which can be faster than the SAS sort. It is also multi-threaded to make use of multiple processors. SyncSort was also available in SAS 8 and later for z/OS and UNIX. You need to use the SORTPGM= option to select SyncSort, because the SAS sort is usually the default. You should always check the actual default for your installation of SAS, as this could vary between operating systems and releases of SAS, or be set by your SAS administrator.
- In SAS®9, the standard SAS sort is multi-threaded and performs very well on multi-processor machines.

```
Proc sort data=x threads ;
  By y ;
Run ;
```

- Use TAGSORT where the BY variables combined length is short compared to observation length, and data set is huge. I have had cases where I was sorting very large data sets and a plain PROC SORT took a long time to run, but using TAGSORT cut the time down to less than half.

```
Proc sort data=x tagsort ;
  By y ;
Run ;
```

- You usually don't need the previous order maintained within the new order, so specify NOEQUALS.

```
proc sort data=data-set NOEQUALS ;
  by y ;
run ;
```

- Allocate more sort work data sets to improve sort efficiency (if you are using z/OS).

```
Options sortwkno=6 ;
```

- Reduce observations sorted by using a WHERE clause.

```
proc sort data=xxx(where=(price>1000)) out=yyy ;
  by y ;
run;
```

- Use all available memory for sorting.

```
options sortsize=max ;
```

- If data is grouped, but not sorted, then use NOTSORTED to avoid the need to sort.

```
proc print data=calendar ;
  by month NOTSORTED ;
run;
```

- If external data (perhaps coming from another database via an import) is pre-sorted, then tell SAS it is sorted in a particular order.

```
Data new(SORTEDBY=year month) ;
   Set x.y ;
run ;
```