Paper 37

# Keeping Your Data in Step - Utilizing Efficiencies

Michael G. Sadof, MGS Associates, Bethesda, MD


## ABSTRACT

In research and in business we typically read and manipulate large amounts of data. Analyzing and processing vast quantities of data can not only be a chore but it can be a resource hog. The SAS® System provides us with the tools to turn that data into information very easily and efficiently. This paper will focus on processing large data sets and explore the ways in which we can use the data step effectively to make the best use of system resources. It will explore topics such as:

- Keep, Drop, and Length statements
- Efficient Merging
- Faster Sorting
- Indexing
- Compressing
- Benchmarking of SQL, vs. Sort, vs. Merge
- Manipulating workspace
- Summarizing
- Reformatting

By utilizing these efficiency techniques you can make the leap from a novice to an expert SAS programmer with ease.


## INTRODUCTION

Efficiency in programming has traditionally been defined as the optimization of space and time. These two quantities are generally inversely proportional so that minimizing the use of space may take more time and vice versa the minimization of time or the increasing of speed may utilize more and maybe unattainable space. I propose that there is another quantity to optimize when writing efficient programs and that is programmer time. Today computer resources are not only defined by CPU clock ticks but by the amount of resources (mainly computer programmers) that are required to develop and maintain a system. This paper shows some efficiency techniques involving the optimization of all three of these resources.

### Space Saving Techniques

When processing large amounts of data whether in research or business we are usually constrained by the amount of space available. Disk space is usually very precious either in a client server or mainframe environment. Your coding practices should be cognizant of this fact and conserve space whenever possible. It's good to get in the habit of space conservation even if the program is not pushing the limits of space.

### Length

One of the simplest and most important space saving techniques is the use of the LENGTH statement. SAS stores numbers differently on the various operating systems but generally stores numbers in floating point binary. The default length of a numeric variable stored in a SAS data set is 8 bytes. In these 8 bytes SAS can store a number with up to 16 significant digits. The largest number that can be stored in a SAS variable of a particular length and the number of significant digits varies between operating systems. What this means for us practically is that if we know approximately how large of a number we are working with we can save space by allocating shorter lengths with the LENGTH statement. If you are working with large budgets reports that may go into the billions and want to keep precision down to the penny you may want to use the full 8 byte storage length. But if you are scoring tests or evaluating surveys with only several million records or primarily using integer values you can significantly reduce your storage requirements with the appropriate use of the LENGTH statement. Unless space is at an absolute premium I recommend the use of the full 8-byte length for numeric variables that will be used in calculations. However for certain types of variables the LENGTH statement is a must. For instance, you can safely store a SAS date up to the year 5000 with a length of 4. This will reduce by half the storage requirements of date variables.

```
DATA ONE;
LENGTH MYDATE 4 ;
MYDATE=TODAY();
```

Another example may be coded data that can range from 1-1000. There is no need to save in 8 bytes when 3 or 4 bytes will be sufficient. Consult the manual for you particular operating system to determine the exact numbers that each length can accurately represent.

### Character Values

The use of character values rather than numbers for categorical data can significantly reduce the space requirements for a program. A single character takes up 1 byte of storage space, while even the shortest length number takes 2 or 3 bytes depending upon the operating system. So for any categorical

data like the type of survey, gender, or the particular type of loan, use character variables. You can use a FORMAT statement to quickly convert between character and numeric variables but be careful because extensive conversions in a large data set can cost you time. In the following example the variable score is numeric. This PUT statement will create a character variable with the length of 2.

```
SCR=PUT(SCORE,$2.);
```

With this technique you can conserve a tremendous amount of space by recoding the scores to character values. Even if you will need to use these numbers in calculations you might want to store them as characters and then convert back to numeric when calculating. This brings us back to our optimization model of time and space. Many techniques that save space will cost us more in time.

**Drop or Keep**

The use of DROP or KEEP statements when reading in a data set can certainly reduce the space requirements but can also significantly reduce the amount of time required to process a data set. It is simple to use as a data set option when reading SAS data sets.

```
DATA FOUR;
     SET THREE(KEEP=A B C);
DATA FIVE;
     SET FOUR (DROP=A);
```

It is preferable to use the KEEP= data set option rather than a KEEP or DROP statement in the data step because in this way SAS only creates space in the program vector for the variables you are using rather than reading in the whole data set and then dropping the variables that are not being used. The KEEP= or DROP= options are equally efficient and it is only a question of style which you should use. Use the KEEP or DROP statements as soon as you first read the data, thus reducing the data you need to process in subsequent steps. A similar technique should be applied to external data sets and will be described later.

**Delete Data sets**

Typically when processing data you use many data steps, merges and PROC SORT's. You can save space by deleting data sets no longer needed by the program. For instance the following code shows two data sets that are merged and then no longer needed separately. You may use PROC DATASETS to delete the unneeded data sets.

```
DATA SIX;
     MERGE SEVEN EIGHT;
  BY TYPE;
PROC DATASETS;
     DELETE SEVEN EIGHT;
```

**External SAS Data Sets**

When data sets are extremely large it is sometimes hard to find space in the SAS workspace for processing. SAS has the ability, under every operating system, to use external SAS libraries for temporary storage or temporary workspace. Your system administrator has set up a default directory or an area where SAS will place temporary files. The size and location of this area can be modified or moved. On the mainframe the SAS environment is specified in the SAS system procedure through JCL statements. These specifications can be overridden by the user with simple JCL statements. MVS allows us to increase the amount of workspace through several methods depending upon your site. Generally by coding a JCL statement similar to the following you can change the default workspace for your job. Please consult your systems manual or systems administrator for exact coding since many installations have site specific methods of increasing workspace.

```
//WORK DD UNIT=SYSDA,
//        SPACE=(CYL,(500,500))
```

or

```
 //   EXEC SAS,
 //        WORK='500,500'
```

Under Windows or Unix the workspace is limited only by the physical amount of free space on your disk drive. The `config.sas` file specifies a directory where your temporary workspace will reside. You can change this directory to a drive with more free space by editing the `config.sas` file and saving your own copy. You can then start up SAS by pointing to your private configuration file with a command something like this:

```
SAS -CONFIG C:\SAS\MYCONFIG.SAS
```

Please note the actual command syntax may vary with each operating system. When your program requires more internal workspace than can be made available in the SASWORK directory you have another option and that is to use external temporary SAS libraries. Rather than trying to fit all your data into the one SASWORK directory you will allocate additional libraries preferably on different disk drives. One way to do this is to build (either with JCL or SAS code) temporary libraries that can be used as additional workspace. This can be done in MVS, UNIX, or Windows. The coding is slightly different under MVS but the results are the same. The LIBNAME statement will create a library on a particular volume. You can create multiple temporary workspaces and then move data between

them as you are processing rather than trying to place all the data in the single SAS workspace. This technique will work on all operating systems with minor changes. Consider the following code:

```
/* allocate two temporary libraries on */
/* different disk volumes              */

LIBNAME TEMP1  'C:\TEMP';
LIBNAME TEMP2  'D:\TEMP';

/* under mvs it might look like this

LIBNAME TEMP1 'HIGH.LVL.QUAL.DATA'
DISP=(NEW,CATLG)
SPACE=(TRK,(5000,100);
*/

/* Sort very large data set: one and   */
/* Store in temporary library since    */
/* there is not enough space in saswork */

PROC SORT DATA = ONE
          OUT = TEMP1.ONE;

/* Now merge another data set with ONE  */
/* and place into a second temporary    */
/* workspace                            */

DATA TEMP2.ONE;
     MERGE TEMP1.ONE
           SIX(IN=OK);
BY TYPE;
IF OK;

PROC DATASETS LIB=TEMP1;
DELETE ONE;
PROC DATASETS LIB=WORK;
DELETE ONE;
```

In this example we allocate space for two temporary libraries. Please note that they should be placed on different volumes so that they will not compete for the same space. The SORT procedure sorts the data set "ONE" from the common SASWORK library to a temporary library temp1. We then merge this data set in the temporary library with another data set (six) and create another temporary library temp2. In this manner you do not need enough space on the c: drive for two copies of the data set. When we are finished merging we can release the space on drive c: by deleting the temporary data set in TEMP1 and in WORK. Remember when sorting in place without the `out=` option SAS requires double the amount of space in the data set. In the following example the workspace would have to be large enough to hold two copies of the data set named happy.

```
PROC SORT DATA = HAPPY; RUN;
```

If you do not have enough space on one volume for two copies of the data set then you must use the `OUT=` option and point it to another volume as in the example. Using this technique you can juggle your data sets back and forth between two temporary libraries without needing an extremely large disk. Individual operating systems may have size limitation for a single data file and with very large data sets you may run into these limitations.

**Compress**

SAS has a built in compression function that can save space and time. Its simple to use but it does not always save space. It is designed to represent repeating bytes in a data set as a string of no more than 3 bytes. For instance a string of 20 blanks might be represented by something like 20*' ' . This technique will require less storage for data sets with fields of sparse data like addresses. Generally we set the length of an address or name to 20-40 bytes but most records will have many blanks. The `COMPRESS=YES` option will reclaim that space for us and sometimes even speed up operation because even though it takes more CPU to pack and unpack the data it takes less IO to read the data from the direct access storage device. I will describe an example run on the windows platform. It is a typical database of survey information and test scores containing names, addresses and scores for 250,000 respondents. Creating, reading and doing a quick frequency on the data set uncompressed took 132 seconds but only 62 seconds compressed. Also the data decreased in size by about 58% from 160 meg to 67 meg. A sort on the same data set compressed took 177 seconds but uncompressed it took 200 not a very significant savings. In other instances, however, when the data set is dense (with little or no repeating characters) a compressed data set can take more space and require more time to process. You will have to do you own experimentation on your data but please do consider using compress. The coding for the compress is a data set option and can be used anywhere data set options are permitted in an output specification. Several examples follow:

```
DATA THREE (COMPRESS=YES);
SET ONE;

PROC SORT DATA=A.DATA OUT=B.DATA
(COMPRESS=YES);

PROC SUMMARY;
VAR SCORE;
OUTPUT OUT=SUM1(COMPRESS=YES) SUM=;
```

You can also specify "`COMPRESS=YES`" as a system option and then every data set created in the job or session will be compressed but since some data sets take up more space when they are compressed I generally control the compression in the data set option. A note of caution: you will not be able to access a compressed data set with random access

techniques such as the POINT= option in the set statement.

**Time Saving Techniques**

Generally the time spent reading and writing data is responsible for most of the time spent in a SAS program. Input/Output (I/O) operations are generally slower than CPU calculations and reformatting. It is therefore always good technique to reduce the number of times you need to read or pass a data set. If you can combine data steps it is helpful.

**Reduce the Number of Data Steps**

You can always improve the speed performance by eliminating unnecessary data steps.
Consider the code

```
DATA THREE;
     MERGE ONE TWO;
BY TYPE;
X=Y*3;
Z=14*Q;

DATA FOUR;
     SET THREE;
L = X + Z ;
```

You can improve the speed of this program by eliminating the 'data four' step altogether and do the final calculation in the 'data three' step. Although this is a simple example as you programs grow it is not always easy to tell where data steps can be combined. Remember, however, that passing a data set twice will use a lot more time.

**Read Only What You Need**

When reading from external data files select records as you input them rather than reading the whole data file in at once and then selecting records. You can read a field in the middle of a data record and hold that record with the trailing @ sign while determining if you need that record.

```
DATA LARGE:
INFILE MYDATA;
INPUT @15  TYPE $2. @ ;
IF TYPE IN ('10','11','12') THEN
   INPUT @1 X  $1.
         @2 Y  $5. ;
```

**Use SAS Data Sets**

SAS reads and processes data more efficiently in SAS data sets rather than in external files. Read external files in to SAS as quickly as possible and keep them in SAS format even if you have to save a file externally. SAS will be able to process the data faster next time you need it.

```
LIBNAME SASOUT 'C:\TEMP\MYDATA';

DATA SASOUT.SAVE; INFILE MYDATA;...
```

You can use this style to produce SAS data sets that others will be using and eliminate the need for subsequent users to read the file with the infile statement.

**Subset Data**

SAS has the ability to create multiple output data sets in a single data step. If you are processing a large data set you can significantly reduce the I/O time by reading the data set only once and creating multiple subsets as you go.

```
DATA MALE FEMALE;
     SET POPULATION;
IF SEX='M'  THEN OUTPUT MALE;
            ELSE OUTPUT FEMALE;
```

Incidentally the use of the ELSE statement will reduce CPU time because once the first condition is met SAS does not process the ELSE condition. By using two if statements you force SAS to process both statements even if the condition is met in the first statement. This is especially helpful when processing multiple conditions with many ELSE statements as follows:

```
IF AGEGROUP=3 THEN DO;...END;
ELSE IF AGEGROUP=2 THEN DO;...END;
ELSE IF AGEGROUP=1 THEN DO;...END;
```

If you expect there to be more respondents with agegroup=3 then put that condition first since SAS will not have to process the additional 'ELSE IF' statements after the first condition is satisfied.

Subsetting of data can also be accomplished with the WHERE clause and this is more efficient than using the IF statement. In this fashion only the data that is needed is passed into the program vector and I/O time is reduced. It will also reduce the space requirement for the resulting output data set. Here are three time comparisons for selection of approximately 40,000 records out of 500,000 records in the uncompressed SURVEY data set. Please note that the time differences seen here are not significant and vary from run to run.

```
/* time=65.5 secs */
DATA ONE; SET SURVEY;
IF TYPE ='A';
RUN;

/* time: 65.8 secs */
DATA TWO; SET SURVEY;
WHERE TYPE='A';
RUN;

/* time=66.9 secs */
DATA ONE;
SET SURVEY (WHERE=(TYPE='A'));
RUN;
```

We will see significant differences in the performance of IF statements and WHERE clauses when we discuss indexing. A good rule of thumb however is to subset early and often. For instance if you plan to summarize data with the PROC SUMMARY do it a soon as possible and add calculations, formatting and labels after the summary is done. In that way you will be performing calculations on fewer observations and your storage requirements for the output data set will also be reduced.

**Indexing**

Indexing is a method by which SAS creates an associated file of the key fields and stores them in such a way that it can access observations quicker and more efficiently under certain circumstances. Data sets do not have to be sorted to create an index. Let's do some benchmarking for the use of indexes. For our survey data set of 500,0000 observations we create an index using PROC DATASETS as follows:

```
PROC DATASETS LIB=MYDATA;
MODIFY SURVEY;
INDEX CREATE YEAR;
```

When we look at the times for the same subsetting code that was run above we see startling differences. Performance is improved when using the WHERE clause on indexed data sets.

```
/* time=65.13 secs */.
DATA ONE; SET SURVEY;
IF TYPE ='A';
RUN;

/* time: 15.75 secs */
DATA TWO; SET SURVEY;
WHERE TYPE='A';
RUN;

/* time: 15.82 secs */
DATA ONE;
SET SURVEY (WHERE=(TYPE='A'));
RUN;
```

The problem with using indexing is that the indexing is not carried down to subsequent data sets or maintained when a data set is created from an indexed data set. SAS will maintain the index structure when copying the data set with PROC COPY or PROC DATASETS but will loose the index when using the DATA and SET combination of statements. A good rule to follow is that if you are going to use a large sorted data set multiple times it is worth creating an index. If you are constantly resorting or accessing with different variables then the time it takes to reindex will outweigh the time it takes to sort and merge.

**PROC SQL**

Significant performance improvements can be seen when using PROC SQL on indexed data sets with the WHERE clause. In the following example the data set SURVEY is 250,000 records out of our survey sample indexed by SSN and the DRIVER file is a 1000 record file sorted by SSN. The merge step step takes approximately 5 minutes and 37 seconds. Then SQL step takes 3.35 seconds while the indexing with PROC DATASETS takes about 1 minute 30 seconds.

```
/* create index on ssn */
PROC DATASETS;
MODIFY SURVEY;
INDEX CREATE SSN;


/* typical merge logic */
DATA ONE;
MERGE SURVEY
     (KEEP=TYPE UNIT SSN)
       DRIVER(IN=OK KEEP=SSN);
BY SSN;
IF OK

PROC SQL; /* SQL LOGIC */
CREATE TABLE ONE AS
      SELECT A.TYPE
            ,A.UNIT
            ,A.SSN
      FROM SURVEY A
          ,SSN    B
      WHERE A.SSN=B.SSN;
QUIT;
```

**Formatting**

It is much more efficient to use PROC FORMAT to recode variables rather than the IF-THEN construct. PROC FORMAT has many forms and can be used to group a range of values. The following simple example shows how to recode and group variables. Use this code:

```
PROC FORMAT;
VALUE GRP
1-10='1'
11-20='2'
21-30='3';
AGEGRP=PUT(AGE,GRP.);
```

rather than:

```
IF  1<=AGE<=10 THEN AGEGRP='1';
ELSE IF 11<=AGE<=20 THEN AGEGRP='2';
ELSE IF 21<=AGE<=30 THEN AGEGRP='3';
```

**Maintenance Efficiencies**

The optimization of time and space parameters are helpful in processing large amounts of data but sloppy, confusing code can negate any time and space savings. SAS programs should be well

documented. It takes only a little time to place some explanations or notes in the code but it will save hours of time when it comes to maintaining the code later. Even if you are the only one using the code it will help you remember exactly what was being done and why. Comments should be placed neatly and consistently throughout the code. A program documentation block should be inserted at the beginning of each program describing the location, purpose, creation date and modification dates of the program.

Use variable and data set names that are meaningful to you or to the business area. Names should be as descriptive as possible within the 8 character limitation. Avoid names that can be construed as SAS commands like: PROC1, CANCEL, PROCSORT. Although they may be valid SAS variable or data set names they will be confusing.

Neatness counts! Try to write your code with reasonable indents, liberal spaces, and appropriately placed comments. Organize similar types of statements together. Place all length and keep statements at the beginning of the data step. Place label, format and drop statements at the end of the data step. Be consistent. Try to keep the code clear with organized indentations and spaces between data steps. It really makes it easier to debug and to review code that has been prepared in that way. The general rule is to indent the code within IF-THEN, DO-END and %MACRO-%MEND constructs as follows:

```
IF X>10 THEN DO;
        TYPE='LARGE';
        Y=X*X;
        OUTPUT;
        END;
IF Z=25 THEN DO;
        ...
        END;
```

These suggestions are, of course, just a matter of style but by developing your own consistent style you will find it easier to code and maintain systems.

**Program Flow**

Try to lay out your approach ahead of time by drawing a flow chart. The flow chart need not be at the most detailed level; a general design will help. Then try to follow the flow diagram (modifying when necessary) as you are building your program. In this way you will be able to avoid sorting data sets twice or reading more fields from a big file than you really need. Try to keep your flow design simple as shown:

1. Read Data file 1 (index on key fields: SSN)
2. Read Data file 2 (extracting only the fields needed)
3. Sort by SSN
4. Merge data file 1 and Data file 2 by SSN creating data file 3
5. Delete data file2 (no longer needed)
6. Summarize data file by school
7. Calculate some statistics
8. Report by school

If the program is complicated you may want to create a more detailed design before proceeding with the program. You will find it much easier to build streamlined code from a program design or flow chart. A straightforward program design will do as much for efficiency as fancy algorithms or procedures that will shave off only seconds.

Label variables in you final output data set and other data sets that are used for reporting. It takes a little more effort while coding but this information will provide information to the business users, other programmers, or even to yourself when you get back to the program several months from now. The PROC CONTENTS is a wealth of information about the data set including size, number of records, and variable labels. Use the contents for determining where savings can be made in terms of lengths of variables or unneeded variables.

When building code that will be used and reused, whether in a user or an ISS production environment it is a good practice to place strategic PROC PRINT's, PUT's and PROC CONTENTS' in the code. This will enable problem tracking without having to rerun the job. You can use PROC PRINTTO to separate business reports from debugging reports or you can write to the SASLOG. I am not a fan of overdoing it with a PROC PRINT after every data step since it can become confusing. During the debugging phase you may want to be more liberal but after the code is tested you may want to remove all but a few well placed debugging prints.
You should create a PROC CONTENTS of the final data set that is produced. This will serve as documentation and, unless the data set is written to tape, it will describe the number of records and the space requirements.

**CONCLUSION**

In this paper I have tried to summarize what I feel are the most important space and time saving techniques. These are by no means exhaustive and one could certainly devote a whole paper to each topic.

Although computer storage is now less expensive and more plentiful it is still a valuable resource. It is important, and sometimes necessary to write SAS programs that optimize the use of space. We have seen that one of the easiest techniques is the use of the LENGTH statement. The proper use of KEEP and DROP statements can also save much space. It is good to get in the habit of only keeping what is needed and setting the length on all variables except for calculated fields. CPU time can be reduced by the use of FORMAT statements rather than the IF-THEN construct. It is also important to prepare programs with the proper organization and flow avoiding wasted steps. A straightforward approach to program design will save a lot of headaches. I/O and CPU time can be reduced by the proper use if indexing, compressing data sets, and the appropriate use of the WHERE clause. PROC SQL seems to improve performance if you are familiar with its syntax. When writing SAS programs one of the most important efficiency techniques should be program clarity. If a program is not understood easily it can not be maintained, modified or improved. Development time can be reduced by charting out the flow of a system before coding, and documenting while coding. Almost every program can be tweaked for efficiency to conserve valuable resources not the least of which is your time.

**REFERENCES**

SAS Institute Inc., SAS® Programming Tips: A Guide to Efficient SAS Processing, Cary, NC: SAS Institute Inc., 1990

SAS Institute Inc., SAS® Companion for UNIX Environments: Language, Version 6, First Edition, Cary, NC: SAS Institute Inc., 1993

SAS Institute Inc., SAS® Companion for the MVS Environment, Version 6, First Edition, Cary, NC: SAS Institute Inc., 1990

SAS Institute Inc., SAS® Companion for the Microsoft Windows Environment, Version 6, First Edition, Cary, NC: SAS Institute Inc., 1993

SAS Institute Inc., SAS® Language: Reference, Version 6, Cary, NC: SAS Institute Inc., 1990



The author may be contacted at:

Michael G. Sadof
MGS Associates, Inc.
6203 Yorkshire Terrace
Bethesda, MD 20814
(301) 530-2353
msadof@ix.netcom.com