# Efficiency Techniques in SAS® 9

Stephen Sloan, Dawson D R

## ABSTRACT

Using space and time efficiently has always been important to organizations and to programmers in general and to SAS programmers in particular. We want to be able to use our available space without having to obtain new servers or other hardware resources, and without having to delete variables or observations to make the SAS data sets fit into the available space. We also want our jobs to run more quickly both to reduce waiting times and also to ensure that scheduled job streams finish on time and that successor jobs are not unnecessarily delayed. Internal mainframe billing algorithms have always rewarded efficiency. As we move toward cloud computing efficiency will become even more important because the billing algorithms in cloud environments charge for every byte and every CPU second, putting an additional financial premium on efficiency.

Sometimes we are in a hurry to get our jobs done on time, so we don't initially pay attention to efficiency, sometimes we don't know at the start of a project how much time and space our jobs will use (and the important time is the time allocated to our assignment), and sometimes we're asked to go into existing jobs and make changes that are seemingly incremental but can cause large increases in the amount of space and/or time that is required. Finally, there can be jobs that have been running for a long time and we take the attitude "if it ain't broke, don't fix it" because we don't want to cause the programs to stop working, especially if they're not well-documented.

With a reasonably good knowledge of SAS® Base, there are things we can do that can help our organizations optimize the use of space and time and run more quickly without causing any loss of observations or variables and without changing the results of the programs.

## INTRODUCTION

There are many areas where we can make our programs run more efficiently and use less space by reviewing the code and inserting appropriate changes that reduce the space required for permanent and temporary SAS data sets, reduce the time required for the calculations, or do both at the same time. At other times there will be potential tradeoffs that need to be evaluated based on the data itself, which leads to an important criterion for many of the methods: **KNOW YOUR DATA**. If the data is unfamiliar to you, you can run some exploratory procedures using PROC FREQ, PROC SUMMARY, and other techniques to learn more about your data.

The efficiency techniques in this paper can be categorized as:

- System-level options
- Sorting options
- The lengths of numeric variables
- The lengths of character variables
- Techniques to reduce the time and space used by PROCs and DATA steps

- Other programming techniques to reduce the space and/or time required
- Using macros to evaluate and possibly reduce the space when there are too many variables to work with each one separately.

## SYSTEM-LEVEL OPTIONS (IN THE OPTIONS STATEMENT)

- Use the COMPRESS= option to reduce the size of SAS data sets. COMPRESS=YES or CHAR reduces the size of character values by compressing repeated occurrences of the same character. This can be especially effective with character variables where some of the observations have shorter lengths than others and thus have many blanks at the end of the record. COMPRESS=BINARY gives greater compression than COMPRESS=YES because it produces variable-length records by shortening all instances of repeating bytes. At times COMPRESS=BINARY has provided me with a 90% reduction in space. Because COMPRESS=BINARY is more complex the compression and decompression could slow down the program, it is only recommended for large SAS data sets.
- Specifying REUSE=YES allows SAS to add observations where space is available during the execution of a program, even if the space has already been used during the program. Even though the unused space is returned after the program is completed, using available space during the program can prevent the operating system from having to swap data in and out of memory when available space is tight
- MSGLEVEL=I can be valuable for the diagnostic information it provides, but MSGLEVEL=N reduces the size of SAS LOGs. If programs require further analysis, you can always rerun the programs using MSGLEVEL=I to get additional information, update the programs as required, and then restore MSGLEVEL=N.
- Similar to the MSGLEVEL option, OPTIONS NONOTES can suppress printing and keep the SASLOG smaller, and you can switch to OPTIONS NOTES if the additional information is required for diagnostic purposes.
- Using OPTIONS ERRORS=0 will suppress information about specific observations in which errors have occurred (for example dividing by 0) and will help keep the SAS LOG file compact. You will still get the message that errors have occurred, even though the specific observations won't be highlighted.
- Using the NOMPRINT and NOMLOGIC macro options will suppress output from macro executions, and thus will reduce the size of the SAS LOG file. As with the MSGLEVEL option, the MPRINT and MLOGIC options can be used when needed to trace macro processing and then can be turned off again when the program is put into production.

## SORTING OPTIONS

Sorting variables can take up a considerable amount of time and use a large amount of sorting workspace. There are SAS PROC SORT options we can use to reduce the space we use. There are also times where PROC SORT might not be necessary if we know the data and can strategically decide where to use it.

- The TAGSORT option will only bring the BY variables into the buffer to be sorted. It will then sort the BY variables and bring in the rest of the variables after the sorting is done. This will reduce the sort workspace used; if the BY variables represent a

small part of the dataset, TAGSORT will lead to a considerable reduction in workspace usage during the PROC SORT. However, since the data is being brought in twice, it could lead to an increase in processing time. Even in this instance, the processing time could go down if the space is not available and the data is being swapped in and out when TAGSORT is not used.

- The PRESORTED option will check whether the SAS data set is already sorted and PROC SORT will not run if the data set is already sorted, saving time and saving sort workspace. Since this adds the step that causes the system to check the dataset, it should only be used if we're not sure whether the dataset is sorted. If we know the dataset is sorted, we don't need to run the PROC SORT at all. If we don't have any reason to think the dataset is sorted, we should just run the PROC SORT without checking via PRESORTED.

- If a SAS dataset is already sorted the way we want it before we run a PROC with a BY variable or before we use the FIRST. and/or LAST. features in a DATA step, we don't need to run the PROC SORT. As mentioned before, **KNOW YOUR DATA** is very helpful when we want to make the programs more efficient.

- When using SAS PROC SUMMARY or similar statements, using a BY variable is more efficient than the CLASS statement because the CLASS statement brings the entire data set into the buffer, while using a BY statement causes PROC SUMMARY to operate on the dataset for each distinct value of the BY variables. If we are sure that the dataset is sorted the way we want it, we can just use the BY statement and save space and time. However, if the data set needs to be sorted, then the combination of the PROC SORT and the PROC SUMMARY with the BY statement could take longer than using the PROC SUMMARY with the CLASS statement, although it would still use the space more efficiently in the PROC SUMMARY.

- If you're just using SORT and MERGE to assign text values to lookup keys (for example assigning names to social security numbers), you can use PROC FORMAT with the CNTLIN feature to create a format within the program. This will allow you to use the format you created wherever required without doing SORT and MERGE and without creating an additional variable. The formats that are produced in this manner use the binary search algorithm and are very efficient in looking up the formatted value.

## LENGTH OF NUMERIC VARIABLES

The default length for SAS numeric variables is 8 bytes. However, if all of the values of a numeric variable are integers, it is often possible to reduce the size of the variable. The chart below shows the required lengths of numeric variables whose maximum absolute value is less than certain values. For example, if the maximum absolute value of a variable is 10,000 and all of the values are integers, the variable can have a length of 4 instead of 8. This can lead to a significant savings in space.

The SAS function for the absolute value of a variable X is ABS(X). The value of the numeric variable X is an integer if X=INT(X). To test every value, you can set a flag:

IF X=INT(X) then FLAG=1;
ELSE FLAG=0;

Then run a PROC SUMMARY MIN on FLAG. If the minimum value is 1, then the variable has all integer values. In this case, you can determine from the chart below whether the variable requires less than 8 bytes and then you can reset the length of the variable using the LENGTH statement.

The chart below can be found in http://support.sas.com/documentation/cdl/en/hostwin/63285/HTML/default/viewer.htm#numvar.htm.

***Significant Digits and Largest Integer by Length for SAS Variables under Windows***

| Length in Bytes | Largest Integer Represented Exactly | Exponential Notation | Significant Digits Retained |
|---|---|---|---|
| 3 | 8,192 | $2^{13}$ | 3 |
| 4 | 2,097,152 | $2^{21}$ | 6 |
| 5 | 536,870,912 | $2^{29}$ | 8 |
| 6 | 137,438,953,472 | $2^{37}$ | 11 |
| 7 | 35,184,372,088,832 | $2^{45}$ | 13 |
| 8 | 9,007,199,254,740,992 | $2^{53}$ | 15 |

Looking at the chart above, we can see that the smallest length of a numeric variable is 3. However, if we look at the MIN and MAX of an all-integer numeric variable X in the data set DS, and all of the values are between -9 and 99, we can change X to a character variable and get a length <3. The following SAS code gives an example:

```
LENGTH X $2.;  /* PUT THIS BEFORE THE SET STATEMENT */

SET DS (RENAME=X=Y);

X=Y;

DROP Y;
```

Finally, if all of the values are missing, you might want to delete the variable to save even more space. This would depend on the specific requirements of your data and your organization. If you want to test for all of the values being missing, you can run the SAS code PROC SUMMARY MIN MAX. If the maximum value = . (missing), then all of the values of the variable are missing. If the minimum value is non-missing, then there are no missing

values in the data set for that variable.  If you want to delete a variable with a certain percentage of the values being missing, you can get the percentage of observations in the data set with the variable X having a missing value by running the following SAS code:

PROC FREQ DATA=DS; TABLES X / MISSING OUT=MISSING(WHERE=(X=.)); RUN;

The variable PERCENT in the output data set MISSING has the percentage of missing values.

## LENGTH OF CHARACTER VARIABLES

Many character variables do not use all of the space allotted to them, especially when SAS has pulled them in from other systems like Oracle or Excel or Teradata, where SAS allows long variable lengths to ensure that no characters are lost due to lack of space in the variable.  We can test the required length for a specific variable by using the SAS LENGTH function to get the LENGTH of each value, using SAS PROC SUMMARY MAX to get the largest required length, and then using the SAS LENGTH statement to reset the length of the variable.

If every value of a character variable contains only digits, and the length of the variable is >=4, we can reduce the length of the variable by converting it to a numeric variable and then re-setting the length using the chart referenced earlier in this paper.  Since the minimum length of a numeric variable is 3, this method only helps if the length of the variable is >=4.

To see if a character variable is all-digits, use the following SAS command to remove the digits:

Y=COMPRESS (X,’ ‘,”D”);

Then, if X^=’ ‘ and Y=’ ‘ then X is all-digits.

Now set a flag.

IF X^=’ ‘ AND Y=’ ‘ THEN FLAG=1;

ELSE FLAG=0;

If we run PROC SUMMARY MIN and the minimum FLAG value=1 then all values of the variable are all digits.

Looking at the above chart, if the largest absolute value of the numeric value of X=1500 then X has 4 characters (5 characters if X=’-1500’) but will have 3 digits if we set X to a numeric variable using the following code:

LENGTH X 3.;  /* PUT THIS BEFORE THE SET STATEMENT */

SET DS (RENAME=X=Y);

X=Y;

DROP Y;

Finally, if all of the values are missing or if a specified percentage of variables are missing, you might want to delete the variable to save even more space. This would depend on the specific requirements of your data and your organization. You can get the percentage of observations with a value of blank in this variable by using the following SAS code:

PROC FREQ DATA=DS; TABLES X / MISSING OUT=MISSING (WHERE= (X=' ')); RUN;

The variable PERCENT in the output data set MISSING has the percentage of missing values.

## TECHNIQUES TO REDUCE THE SPACE AND TIME USED IN SAS PROCS AND DATA STEPS

- Using DROP, KEEP, and RENAME statements will reduce the number of variables used in SAS data sets
    - The DROP command in the DATA step will DROP variables not needed and the KEEP command will only KEEP the required variables. You should use one or the other on a specific data set. Whichever involves fewer variables will give you more compact programs.
    - If we use the DROP= or KEEP= options in data sets being brought into DATA or PROC steps there is a further savings because the variables won't come into the buffer in the first place. Likewise, if we use DROP= or KEEP= statements when referencing the output SAS data sets, we won't need to go back and drop the unneeded variables later.
    - PROCs sometimes produce additional variables on output data sets that might not be needed. For example, PROC SUMMARY and PROC MEANS produce the variables _TYPE_ and _FREQ_ to aid in later diagnostics. If these variables aren't necessary, we can add {DROP=_TYPE_ _FREQ_} to the statement identifying the output SAS data set and reduce the space required for the dataset and the time required to output the dataset.
    - The RENAME command will change the name of a variable from X to Y more quickly than saying Y=X; DROP X; This can be especially efficient when doing a MERGE or SET of SAS data sets that have the same variable with different names. By using the RENAME= option when the data set is pulled into the SET or MERGE statement, we save the time and space involved in creating another variable and assigning a value to it.

- Using subsetting IF statements and WHERE statements will keep the dataset from having unnecessary observations. Use these commands as early as possible in the DATA step so that the later commands are not executed if they're not required. Furthermore, if we put a WHERE= clause in the data set being brought into the buffer, that will lead to even more savings by preventing the observations from being brought into the data set in the first place. Finally, we can use WHERE= clauses on inputs to PROCs so that we don't have to drag unnecessary observations through the process. If we use WHERE= clauses on the outputs from PROCs we don't have to carry the observations through or delete them later from the output SAS datasets. The WHERE= option can be used together with DROP=. KEEP=, and/or RENAME= to

reduce both the unnecessary variables and the unnecessary observations in a single statement.

- When running a series of IF THEN ELSE commands or using a CASE construct, you should run the most common occurrence first, followed by the second most common occurrence, etc. This is because the search stops as soon as you hit a match, so the instances where the first condition is met take the least amount of time. This is another example of the situation where it is important to **KNOW YOUR DATA** if you want the program to run as efficiently as possible.

- SAS PROC APPEND concatenates the data set you identify as DATA= to the data set you identify as BASE=. It doesn't write out the BASE= data set, it only concatenates the DATA= data set to the BASE= data set, so you will save additional time if you use the data set with more observations as the BASE= data set. Likewise, if you are using SET to concatenate two data sets, and you're not doing any other processing, PROC APPEND will be more efficient than using SET. If the BASE= data set has the same variables as the DATA=data set, but some have different names, you can save time by using the RENAME= option to rename the variables in the DATA= data set.

## OTHER PROGRAMMING TECHNIQUES TO REDUCE THE SPACE AND/OR TIME REQUIRED

- Although the WORK data sets are temporary and are deleted at the end of the program, they still take up space while the program is running. Many people leave SAS EG running for long periods of time and the WORK files accumulate. It would make sense to delete the WORK files and permanent SAS data sets when they are no longer needed. This can be done with SAS PROC DELETE or SAS PROC DATASETS.

- If you're just using the data step to write out a file or set the values in macro variables, you don't need to create a SAS data set that will be deleted later. You can use DATA _NULL_ in these instances and you won't create a SAS data set.

- If you're extracting data from external databases like Oracle, it's best to use the CONNECT statement with PROC SQL instead of using the LIBNAME statement. With the CONECT statement the SELECT statement in your PROC SQL does all of its work within the external database and then brings the selected and possibly summarized or transformed data back to SAS. If you use the LIBNAME statement, all of the data is brought into SAS before being processed and this can take a considerable amount of time and use unneeded space, especially if we only want to use a small number of the rows and columns from the external file. The one drawback to the CONNECT is the fact that, since data is being processed in the external database, we can't use any SAS functions that aren't part of the SQL commands used in the external database, so we need to run a second step after the data has been brought into SAS. However, this additional bit of work usually adds much less time than the amount of time saved by doing the work in the external database while accessing it via SAS.

## MACROS CAN HELP IF THE SAS DATA SET HAS A LARGE NUMBER OF VARIABLES

The methods mentioned above to reduce the size of numeric and character variables can be done by hand if there are a small number of variables in the data set. However, many of the data sets we'll be looking to optimize have hundreds of variables and it would take a long time to run each test by hand. In this case writing a macro is the way to go.

The SAS data set SASHELP.VTABLE contains a lot of metadata about the SAS data sets and is available during a SAS session or while running a SAS program. You can identify the library and the data set name and place the numeric and character variables in separate SAS data sets with the following code where &ds is the data set you want to optimize:

```
DATA numeric (KEEP=_NUMERIC_) character (KEEP=_CHARACTER_);

     SET &ds;

RUN;
```

You can also use SAS PROC CONTENTS and create an output data set to identify the numeric and character variables. The output data set from PROC CONTENTS has a variable named *Variable,* a variable named *Type* that has the values "Num" or "Char", and a variable named *LEN* that has the length of the variable. Whether using PROC CONTENTS or the _numeric_ and _character_ identifiers we now have a list of the variables in the SAS data set that we can parse to get the appropriate lengths and assign them to the variables.

The final step is to concatenate the 4 data sets we created: numeric variables, numeric variables converted to character variables, character variables, and character variables converted to numeric variables. Sometimes blank character variables use less space than missing numeric variables after compression, so we need to do a final check to see whether or not we want to use the variables that have converted between numeric and character.

Michael Raithel, michaelraithel@westat.com, delivered a very interesting talk at SESUG 2024 about a macro he developed called SAS Data Set Missing Value Analyzer.

\\CarsProject\ProgramLibrary\SAS Data Set Missing Values Analyzer.sas

This macro can be used for a number of purposes, one of which is to check for observations with all or most of the values missing. Depending on the organization's policies, these observations could be deleted.

## CONCLUSION

We often run into situations where we need to reduce the size of SAS data sets and we need to reduce the run time of SAS programs. This has always been important when we needed to efficiently exploit the available server space and keep the jobs from running

too long (especially when they run as part of job streams that need to complete by a specific time). Now, with work being done in the cloud, the cloud billing algorithms can cause the organization to incur higher costs if space and time are not used efficiently.

In this paper, I have attempted to point out areas where the efficient use of space and time can be implanted within SAS programs. I'm sure I barely scratched the surface and I'd be happy to hear about other techniques and incorporate them into future releases.

## CONTACT INFORMATION

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

Stephen B. Sloan
Dawson D R (Data Research)
Data Science Senior Principal
Stephen.stephensloan@gmail.com
https://www.linkedin.com/in/stephen-b-sloan/

Stephen B. Sloan has worked in a variety of functional areas including Project Management, Data Management, and Statistical Analysis for Human Resources, Supply Chain, Finance, Marketing, Insurance, Life Sciences, and Manufacturing on behalf of both private and government clients. Stephen has had the good fortune to have worked with many talented people at SAS Institute. Stephen has presented over 100 times at 51 SAS conferences and been published in professional journals. Stephen has a B.A. cum laude with Honor in Mathematics from Brandeis University, M.S. degrees in Mathematics and Computer Science from Northern Illinois University, an MBA from Stern Business School at New York University (1st in his class), and a graduate certificate in Financial Analytics from Stevens Institute.