PharmaSUG 2021 - AP-020

Twenty Ways to Run Your SAS Program Faster and Use Less Space

Stephen Sloan, Accenture

ABSTRACT

When we run SAS® programs that use large amounts of data or have complicated algorithms, we often are frustrated by the amount of time it takes for the programs to run and by the large amount of space required for the program to run to completion. Even experienced SAS programmers sometimes run into this situation, perhaps through the need to produce results quickly, through a change in the data source, through inheriting someone else's programs, or for some other reason. This paper outlines twenty techniques that can reduce the time and space required for a program without requiring an extended period of time for the modifications. The twenty techniques are a mixture of space-saving and time-saving techniques, and many are a combination of the two approaches. They do not require advanced knowledge of SAS, only a reasonable familiarity with Base SAS® and a willingness to delve into the details of the programs. By applying some or all of these techniques, people can gain significant reductions in the space used by their programs and the time it takes them to run. The two concerns are often linked, as programs that require large amounts of space often require more paging to use the available space, and that increases the run time for these programs.

INTRODUCTION

Twenty ways to have your program use less space and time:

1. Use only the variables that you need. DROP and KEEP statements and DROP= and KEEP= SAS data set options will instruct SAS about which variables you need. Using DROP= and KEEP= on the input data sets is more efficient than using them as program statements because they don't bring the unneeded variables into the buffer. The DROP= and KEEP= can also be used on output statements in DATA and PROC steps. Using the DROP= and KEEP= clauses in the PROC saves a step when compared with creating a data set in a DATA step, determining which variables to drop or keep, and then running the PROC. The DROP and KEEP statements can only be used in DATA steps, while the DROP= and KEEP= data set options can be used in both DATA steps and PROCs.

2. Use subsetting IF statements or WHERE statements to reduce the number of observations that are output to the SAS data set. Use WHERE= on the input SAS data sets where possible to reduce the number of observations brought into the buffer. WHERE= can also be used in OUTPUT statements, for example in PROC SUMMARY or PROC SORT. Using the WHERE= clause in the PROC saves a step when compared with creating a data set in a DATA step using WHERE= and then running the PROC.

3. When outputting a SAS data set from a DATA step or a PROC, use KEEP and DROP in DATA steps and KEEP= and DROP= in DATA steps or PROCs to reduce the space requirements. For example, the output from PROC SUMMARY includes two new variables, _TYPE_ and _FREQ_, and these are not often used. You can also use

KEEP= and DROP= in conjunction with a WHERE= clause to further restrict the number of observations in the output data set.

4. Put RENAME= in SET or MERGE statements where possible, or directly in a PROC step. This avoids the need to act on the variable after it has been brought into the buffer in a DATA step. It also can eliminate the need for a separate DATA step to rename the variable before merging with a data set that has the same variable with a different name.

5. Use the LENGTH command to define the length of character and numeric variables. This can achieve a significant reduction in the space used by the program.

6. Numeric variables in SAS data sets have a default length of 8. If the values of the numeric variable are all integers, you can reduce the space by using the following table. The third column refers to the absolute value of the number. Calculate the largest value of the numeric variable by using the MAX option in PROC SUMMARY, check to make sure all values are integers by making sure that the variable's value is the same as the value calculated with the ROUND function, and then, if the variables are all integers, use the table below to determine the smallest length required. 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

Figure 1. Space occupied by numeric variables.

7. Sometimes character variables imported into SAS from other systems, like Oracle or Excel, have very large lengths. You can use the following procedure to get the shortest possible length for your character variable, although you might want to allow room for growth:
    a. Use the LENGTH function to calculate the actual length of the variable in each observation in the data set.
    b. Use the MAX option in PROC SUMMARY to get the largest value of the length.
    c. Use the LENGTH statement to shorten the length of the character variable to the maximum length.

8. Switch variables from numeric to character if they are integers and range in value from -9 to 99. The minimum length for numeric variables is 3, so you can save space if the variable can fit into one or two characters.

9. Switch variables from character to numeric if they are all integers and occupy more than 3 bytes. For example, the number 1234 would occupy 4 bytes as a character variable but item 6 above shows it would only occupy 3 bytes as a numeric variable.

10. Use the options REUSE=YES and either COMPRESS=YES or COMPRESS=BINARY in an OPTIONS statement to save space during the program. However, be aware that the COMPRESS=YES or COMPRESS=BINARY options might increase the amount of time that the program runs. COMPRESS=BINARY saves even more space than COMPRESS=YES but also could have a greater impact on run time.

11. If you have a large data set to sort, using the TAGSORT option with PROC SORT will take up less sort work space, although it could cause the program to run longer. This is because it only brings in the variables in the BY statement for sorting, and then goes back and brings the entire observations into the buffer in the order determined by the PROC SORT.

12. If you have data sets that assign text values to codes, use PROC FORMAT with the CNTLIN= option to create a format from the data set containing the codes and associated text values. Doing this takes less time than doing a SORT and MERGE to create an additional variable in the data set. You can then use the format you created to translate the codes to the text values by using the PUT function. If you already know the values required, using a FORMAT is still faster than a series of IF-THEN-ELSE statements or a CASE or SELECT sequence.

13. Although WORK SAS data sets will be deleted at the end of the program, they occupy space while the program is running. Permanent and WORK SAS data sets that are no longer needed can be deleted while the program is running by using PROC DELETE or PROC DATASETS. This is especially important when using SAS EG because WORK files remain while the session is open, even if the program has finished running.

14. When pulling data from external data bases like Oracle, do as much of the work as possible in the external data base through your SELECT statement or its equivalent. With Oracle, you can use CONNECT in PROC SQL instead of the LIBNAME statement. That way you're not bringing unneeded variables and observations into the buffer. The one downside to this method is that you might have to use two statements instead of one: one statement to process statements in the data base and one statement to use the SAS features or join to other SAS data sets after the data has been extracted.

15. Using PROC APPEND to concatenate two SAS data sets takes less time than concatenating them through a SET statement. Instead of rewriting both data sets PROC APPEND just writes the data from the data set identified by DATA= after the observations in the data set identified by BASE=. To minimize the time involved, use the larger data set as the BASE data set, so you're only copying from the smaller data set.

16. When using PROC SUMMARY or PROC MEANS, use a CLASS statement if the data is not sorted by one of the variables under consideration. This will avoid the time used in a PROC SORT.

17. When using PROCs that allow for a BY statement, such as PROC SUMMARY or PROC MEANS, use a BY statement for variables by which the data set has been sorted. This will take up less space during execution, as the PROC will run separately for each unique value of the BY variable(s). Since the input data set is already sorted, you will not have to run a PROC SORT before using the BY statement in the PROC.

18. If you know the data, then put the most commonly occurring situations at the start of IF-THEN-ELSE or CASE or SELECT sequences. That way, the program will only execute the minimum number of comparisons before moving on to the next commands.

19. If you are not creating or modifying a SAS data set, but are just writing out a sequential data set, use DATA _NULL_ to avoid creating an unnecessary SAS data set.

20. If a SAS data set might already be sorted, and has not gone through a PROC SORT, you could use the PROC SORT option PRESORTED. This will check to see if the data set is sorted by the variables in the BY statement and will not sort it if it is already sorted. Since it involves an extra check through the data set, only use it when you think the data set might be sorted. If the data set has already been sorted with a SAS PROC SORT, there will be a flag, and it will not be re-sorted.

CONCLUSION

When faced with the need to make your SAS programs run faster and/or use less space, there are some guidelines you can follow. This procedure will give you a step-by-step guide to reducing both the footprint and the time taken by the program and will allow you to focus on what you are trying to accomplish with the program, instead of continuously being interrupted by errors due to lack of space or taking too much time.

CONTACT INFORMATION

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

Stephen B. Sloan
Accenture
Data Science Senior Principal
Stephen.b.sloan@accenture.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.