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

Stephen Sloan, Accenture

ABSTRACT

When running 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.

INTRODUCTION

The following 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 SAS Base 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.

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.

```
DATA X;
  SET Y;
  KEEP A B C;
RUN;

PROC SORT DATA=X OUT=Y(DROP=EXTRA);
  BY A;
RUN;
```

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.

```
DATA X;
  SET Y;
  IF FLAG='Y';
RUN;

PROC SORT DATA=X(WHERE=(FLAG='Y'));
  BY A;
RUN;
```

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.

```
PROC SUMMARY SUM DATA=X (KEEP=A B C WHERE=(FLAG='X'));
  CLASS A;
  VAR B C;
  OUTPUT OUT=SUM_OUT(DROP=_TYPE_ _FREQ_) SUM=;
RUN;
```

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.

```
DATA Z;
  MERGE X Y(RENAME=FLAG=A);
  BY A;
RUN;
```

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.

```
LENGTH old_value $ 1; /* Set the length and the character value */
SET ds(RENAME=old_value=new_value); /* Rename the numeric variable */
old_value=new_value; /* This moves the numeric to a character variable */
DROP new_value; /* Clean-up */
```

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, check to make sure all values are integers by comparing the variable's value to 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, Table 1, can be found in http://support.sas.com/documentation/cdl/en/hostwin/63285/HTML/default/viewer.htm#numvar.htm.

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

Table 1. Significant Digits and Larges Integer by Length for SAS Variables under Windows

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.

    ```
    L=LENGTH(A);
    ```

    b. Use the MAX option in PROC SUMMARY to get the largest value of the length.

    ```
    PROC SUMMARY MAX DATA=X;
          VAR L;
          OUTPUT OUT=TEST_LENGTH MAX=;
    ```

    c. Use the LENGTH statement to shorten the length of the character variable to the maximum length. See code snippet after item 5.

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.
   See code snippet after item 5.

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.

    ```
    /* See which character variables contain only digits. */
    no_digits=COMPRESS(var,d);
    length_without_digits=LENGTH(no_digits);
    ```

    Use PROC SUMMARY with MAX to test length_without_digits. If the max of length_without_digits is 0, the variable is an all-digit character variable.

    ```
    /* Convert the all-digit character variables to numeric variables */
    LENGTH old_value 3; /* Set the length of the numeric variable */
    SET ds(RENAME=old_value=new_value); /* Rename the character variable */
    old_value=new_value; /* This moves the character to a numeric variable */
    DROP new_value; /* Clean-up */
    ```

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.

    ```
    OPTIONS REUSE=YES COMPRESS=BINARY;
    ```

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 will 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.

    ```
    PROC SORT DATA=X TAGSORT;
      BY A;
    RUN;
    ```

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.

```
DATA FMTDATA;
         SET X;
         START=A; /* This is the code */
                LABEL=B; /* This is the result */
  FMTNAME='$TR';
RUN;

/* Create the format $TR using the translation defined above */
PROC FORMAT CNTLIN=FMTDATA;
RUN;
```

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. 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.

```
/* The following is faster because it does the work in Oracle:*/
proc sql;
       connect to oracle ( user=oracle_id orapw=oracle_passwd
       path="@t:oracle_server:oracle_sid");
       create table libref.dataset_name as
              select * from connection to oracle
       (select * from oracle_table );
       disconnect from oracle;
quit;
```

The following allows you to access the Oracle table as if it were a SAS data set. It permits you to use SAS functions that might not be Oracle functions, but it takes longer to run.

```
libname db-libref oracle user=scott password=tiger path=oracle-server-
path preserve_col_names=yes /*optional, retain lowercase column names */;
```

15. Using PROC APPEND to concatenate two SAS data sets takes less time than concatenating them through a SET statement. Use the longer data set after BASE= because that data set is not rewritten. 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=.

```
PROC APPEND BASE=LONGER DATA=SHORTER;
RUN;
```

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.

```
PROC SUMMARY DATA=X SUM;
  CLASS A;
  VAR B C;
  OUTPUT OUT=WITH_CLASS SUM=;
RUN;
```

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.

```
PROC SUMMARY DATA=X SUM;
  BY A;
  VAR B C;
  OUTPUT OUT=WITH_BY SUM=;
RUN;
```

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.

```
/* Flag is usually Y*/
DATA RESULT;
       SET X;
       IF FLAG='Y' THEN TEST='SUCCESS';
       ELSE TEST='FAILURE';
RUN;
```

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.

```
DATA _NULL_;
  SET X;
  FILE OUTDS;
  PUT A B C;
RUN;
```

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.

```
PROC SORT DATA=X PRESORTED;
  BY A;
RUN;
```

CONCLUSION

By using the above techniques, people can run their programs more efficiently with smaller footprints. In addition to helping the programmer get more done, the techniques can reduce the overall load on the operating system and thus avoid complicating other programming efforts.

CONTACT INFORMATION

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

Name: Stephen Sloan

Enterprise: Accenture

Title: Data Science Senior Principal

Work Phone: 917-375-2937

E-mail: Stephen.b.sloan@accenture.com