TECHNIQUES FOR EFFICIENTLY ACCESSING AND MANAGING DATA

Steven A. Wilson, Kaiser Permanente Division of Research, Oakland, CA.

ABSTRACT

This paper explores a number of SAS System data management and coding techniques for improving efficiency. After efficiency is defined and a number basic efficiency techniques are summarized, some SAS Version 6.07+ features are discussed from an efficiency viewpoint. These features include PROC SQL, SAS data set compressing, the Stored Program Facility, and SAS Data Views.

INTRODUCTION - Efficiency Defined

Efficient coding is more than being able to code a program that will run the quickest. Efficiency can be broadly defined as completing a task while utilizing the minimum resources.

Since there are many different resources used in completing a programming task, there are many aspects to efficiency. These include elapsed time, CPU time, memory usage, data storage, and I/O operations. Also very important, and too frequently ignored, is the expenditure of programmer time for design, coding, and maintenance. All these factors should be considered when measuring efficiency.

Saving programmer time is the major advantage of the SAS System over all third generation language (3GL) programs. Compared to 3GL programs, the SAS System greatly reduces the expenditure of programmer time while substantially increasing CPU costs. This reduction in programmer time helps avoid the large application backlogs and high program maintenance costs that are endemic in 3GL shops.

To measure efficiency within the SAS System, various SAS System Options can be used to obtain performance statistics. Often used system options are indicated below. Some options are platform specific, as noted.

```
OPTIONS NOTES
 MPRINT SOURCE SOURCE2 /* Log source */
 STIMER FULLSTIMER     /* Usage stats*/
 MSGLEVEL=I            /* PC Only    */
 MEMRPT  FULLSTATS     /* MVS only   */;
```

All of these options utilize resources too, so be sure to use them only when needed.

Efficiency Goals

The efficiency techniques that you use will depend upon your specific goals. Think of efficient programming as being realized via a set of progressive goals, where basic efficiency goals should to be met before progressing to more specialized goals.

This is because the biggest gains in efficiency often result from basic efficient coding, not from highly specialized tricks (although these do have their uses). Efficient SAS coding is developed as you learn how the SAS System works and can anticipate how the SAS System will work with a given set of code.

Some sample goals are listed below:

■ Learn Basic Efficient Techniques

Fortunately, the biggest savings in SAS coding are obtained from the most basic and, usually, intuitive techniques. These techniques are summarized in a SAS Institute manual entitled "SAS Programming Tips: A Guide to Efficient SAS Processing" and are mainly a combination of good programming practice and common sense. Incorporate these techniques into your daily life until they become automatic.

■ Minimize Costs

By determining your costs, you can address issues that directly affect your organization's budget. For example, if your time (including benefits) is valued at $20 per hour and you are charged 25¢ per CPU second, one hour of your time is worth 80 CPU seconds.

This amount of CPU reduction can easily be obtained by making a large, inefficient job more efficient. On the other hand, a huge loss could occur by re-writing an application using techniques that will save only a minimal amount of resources.

Also, if any of your expenses are variable, attempt to take advantage of this variability (eg. lower job costs that may be offered for off-peak job submissions, better response time during the lunch hour).

Recognizing *all* the costs associated with an application allows you to better control these costs.

■ Minimize Utilization of Critical Resources
Data storage may be cheap and abundant at your site, making CPU usage a focus of this goal. Memory constraints may be more of a concern for smaller machines. By identifying critical resources on your platform you can code to decrease usage of these resources. Accomplishing this usually increases utilization of other, less critical or costly resources.

■ Optimize an Application
Is your application a batch job or an interactive system? Speed is especially critical for on-line applications. Does your application use large data sets or small? Very large data sets may require that you attempt to optimize data storage and I/O considerations at the expense of other factors.

Since a great deal of programmer time can be spent optimizing an application, be reasonably certain that the effort will result in the necessary savings.

One good method here is to have experienced SAS programmers periodically review applications coded by novice SAS programmers. More seasoned SAS programmers are easily able to point out inefficiencies and applicable SAS capabilities that the novice many not realize. Not only does this help optimize an application, but the mentoring will make the novice programmer more productive in a much shorter time period.

■ Lower Costs in an Already Efficient Application
By the time you have progressed to this goal, your code will be so efficient that squeezing those last few cents from an application will be difficult. The best situation would be when working with large systems or implementing a new feature of the SAS System.

■ Cooperative Processing & Distributed Data Goals
With the SAS System now widely available on multiple platforms in an organization, we are now faced with not only learning generalized efficiency techniques but also utilizing the relative efficiencies of different platforms in order to achieve our goals.

Thus, while basic efficiency techniques are applicable to all platforms, each of the other goals may be achieved differently for each SAS platform.

While you would ideally like to move as much processing as possible to the cheapest and/or least utilized platform, reliability can become a problem. My experience is that the least utilized platform is usually also the least reliable in operation.

Note that to fully realize this goal, your site must have SAS/CONNECT and SAS/ACCESS installed and operational. Since this is beyond the scope of this paper, refer to Creech & Van Wyk (1991).

Test Data Sets and Testing Platforms

The results of test runs are not presented in this paper due to space limitations. Rest assured, though, that there were hundreds of tests run to establish this synopsis of techniques that showed a significant advantage. Feel free to contact the author for the actual code, results of tests performed, or copies of the handouts which show some results in graphical format.

These tests used 12 different SAS 6.07 data sets, each containing a mixture of randomly generated numeric and character variables. They were run in Version 6.07 on an Hitachi 9021 mainframe under MVS. The characteristics of these data sets are as follows:

Large	150,000 obs	25, 75, and 150 variables
Medium	25,000 obs	25, 75, and 150 variables
Small	6,000 obs	25, 75, and 150 variables
Very Small	1,000 obs	25, 75, and 150 variables

(Record Lengths = 191, 670, and 1340)

The Small, and Very Small data sets were also tested via a LAN based SAS, Version 6.08, on an IBM PC with 8 megabytes of memory under Windows 3.1 with an Intel 80486 and a math co-processor.

This PC configuration is no trivial matter. Version 6.08 averages about 40% faster than Version 6.04 with the aforementioned configuration. However, an Intel 80386 with 4 meg of memory and no math coprocessor takes twice as long to run 6.08 as it does 6.04. A CPU intensive job that ran for 3:07 hours on the 386 configuration was finished in 9:10 *minutes* on the 486 configuration.

As previously mentioned, there will usually be different costs and resources associated with different SAS platforms. Because of this, different statistics were used when measuring efficiency on the two testing platforms.

Mainframe charges are billed back at my site according to CPU usage and tape I/O''s. There are also costs associated with the amount of data storage that is used. Thus, efficiency techniques will be measured according to the usage of these resources.

Costs associated with PC usage are quite different. Assuming that most applications are run from a SAS Display Manager System session, the main concern with this platform is speed of execution. Memory constraints are also a consideration, and, by utilizing a network server, I/O operations become somewhat important too.

The next few sections of this paper simply list techniques that have been shown to be effective. Although I don't go into much detail, my hopes are that this list will pique your interest in a number of new ways that you can then investigate on your own platform with your own data.

Programmer Time - A Big Saver

■ Know Your Data

- Familiarity with your data saves time coding and re-work. A PROC CONTENTS as well as a PROC PRINT of the first few observations are beneficial.

- Uses of the data will impact the design of an application as well as the elements in the data sets. Think beforehand about how your data will be manipulated, analyzed, and stored.

■ Design Before Coding

- Design the flow of data through your application.
- Flowcharting helps.....honest!

■ Develop and Follow Coding Conventions

- Make your jobs easy to read
- Place a descriptive header on each job
- Code only one statement per line
- Use lots of blank lines
- Follow a consistent indentation scheme
- End every DATA and PROC step with a RUN
- Use lots of source code comments

■ Define File References in your SAS Source Code

- LIBNAME and FILENAME statements can create and allocate SAS libraries and external files, eliminating the need for host commands. This keeps job code in the SAS LOG, instead of some host job control log.

- FILE and INFILE statements allow the specification of the full external file name. For example,

```
FILE   'A:\MYDIR\FILE.TXT';
INFILE   'MY.MVS.FLAT.FILE';
```

■ Documentation

- An important but often ignored part of the job. This is especially important if your code will be used or maintained by others, or if the code is tricky or infrequently executed. Be sure you allocate enough time for proper documentation of all data sources and programs.

- Develop and use a consistent documentation style to make your documentation easier to read. Regarding source code comments, I find that a comment box works better than a single line statement. Also, comments of the form **message;* are compiled, and hence more costly, than comments of the form */*message*/*, which are not compiled. However, this is extremely minimal.

■ Develop Job Skeletons / Copy Files

- Having source code files that are designed to be copied and modified saves you from re-typing often used code. These copy files should be expertly coded and documented, allowing you to quickly delete unneeded code, keeping only what is applicable to the task at hand.

For example, when beginning a new program, I may copy an MVS JCL skeleton, an input file specification for an external file, a standard comment box, and a PROC TABULATE outline into the job I am building. This gives a bug-free foundation from which an application may be developed.

■ Create a Set of Flexible Tools

- Develop your own utility library to allow you to quickly and accurately perform routine tasks. SAS macros are especially useful for creating utility libraries (Wilson, 1994).

- Data driven programming and programs that generate other programs are especially useful techniques for applications development. Most medium-to-large applications that I develop use some sort of status data set to control actions to be performed.

Using SAS Effectively

■ Know How SAS Works

- Understand the macro resolution, compile, and execution phases of a SAS job (Henderson, et. al., 1991). Many inefficiencies result from a lack of understanding of the SAS System data management and procedural methods.

- A good knowledge of how the SAS System operates internally allows you greater insight into how to let SAS work efficiently. It also lets you make educated assumptions about how SAS will handle a given set of statements.

■ Know SAS Capabilities

- Familiarize yourself with SAS PROCS, functions, and SAS System Options.

- Aside from experience, a good way to increase your SAS knowledge is to browse SAS manuals and WUSS or SUGI Proceedings.

- Get continual training for you and your staff. The SAS System has grown dramatically in the past few years. Be sure to keep up with the enhancements.

■ Use Character Variables

- Numeric variables are much more CPU intensive than character variables. Only analysis variables need to be numeric.

■ Use a LENGTH Statement for Numeric Variables

- This not only saves space and I/O, but also CPU utilization of some PROCs as well when used to reduce storage space for numeric integer variables. Length of 4 bytes is sufficient for SAS dates while the minimum acceptable length is 2 bytes.

Consult the SAS Companion manual for your platform for a table showing the "Significant Digits and Largest Integer by Lengths".

Lengths for non-integer variables can also be shortened, but beware that precision can be lost if the length is too short. Also be careful when porting data between platforms.

■ Use Formats and Informats

- Saves storage space.
- Saves CPU when used to perform complex assignments.
- Create your own permanent SAS format catalog.

■ Effectively use SAS Libraries

- Be familiar with concepts in Chapter 6 ("SAS Files") of the SAS Language manual. This chapter includes descriptions of the SAS Multiple Engine Architecture, forms of SAS data sets, and SAS data library contents - concepts with which you need to be comfortable.

- Explicitly defining the data engine on a LIBNAME statement saves CPU.

■ Keep the WORK library clean

- Use PROC DATASETS to delete unneeded data sets from the SAS WORK library. SAS works harder when reading from or writing to a full SAS library.

■ Minimize the Work Required from SAS

These are essential tips for efficient programming!

- *Minimize the number of passes through your data.*
- *Minimize the number of DATA steps.*
- *Create multiple data sets with one DATA step.*
- *Do not perform unnecessary sorting.*

■ Use SAS OPTIONS effectively

```
OPTIONS  OBS=10  ERRORABEND;
```

Some SAS options are good for use when developing and testing an application. The above options will prevent a job from using excessive resources during testing. OBS can be set to 0, but I prefer to pass a few observations so analyses can be performed and output generated (NOTE: OBS=MAX must be in effect when a WHERE= data set option or a WHERE statement is encountered) .

```
OPTIONS BUFNO=8;
```

Other options are more specialized and can be used to improve the SAS System's performance with respect to various resources. The above option, while utilizing more memory, was extremely useful for saving I/O as well as CPU on the mainframe. Tests on the PC with differing values showed no improvement over the default value of 1.

Many of the specialized options are host specific and are documented in the SAS Companion manuals, usually in a section entitled "Optimizing Performance" or some similar heading. While these options can be more difficult to use and generally require testing, there can be large savings realized from their use.

■ Use SAS Data Set Options liberally

- SAS data set options can be attached to a SAS data set name at any time in any place.

- Use a KEEP= or DROP= data set option to immediately eliminate unwanted variables when a data set is first referenced.

- Use the WHERE= data set option to realize big savings when a subset of a SAS data set is needed. This avoids having to create a separate data set containing just the subset. However, if a data set is going to be processed repeatedly by WHERE options, it may be more efficient to actually create the needed subset(s) as a separate data set(s).

For example, if running a number of PROC FREQs on various subsets of a large SAS data set, it can be cheaper to code a single DATA step to create all

of the subsets required and then run the PROC FREQs on the created subsets rather that simply passing the large data set to each PROC FREQ with a WHERE= data set option or WHERE statement.

■ Use of Macros
- Although compiling a macro can be costly and learning the SAS macro language is time consuming, I find SAS macros to be invaluable for programming.

- If not using macros, use the NOMACRO option at SAS invocation to stop SAS from needlessly checking for macro code, eliminating the macro resolution phase of a job. Savings will depend upon the amount of code being compiled.

- Macros can be problematic on machines with memory constraints.

■ Testing
- Use `RUN CANCEL;` for incremental testing. This will compile the code in a DATA or PROC step and check for syntax errors without executing the step.

- Another technique for testing a SAS application involves using the SAS macro language to provide intermediate output for verification when an application is executed in the "test mode". For example:

```
%macro job(test=N);
  %if &test = Y %then OPTIONS MPRINT;;
  .....SAS code.....
  %if &test = Y %then %do;
    TITLE3 'Intermediate data set...';
    PROC PRINT DATA=TEMP;
    RUN;
    TITLE3;  /* Re-set title */
  %end;
  .....more SAS code.....
%mend job;
%job;
```

This job can be run in either "test mode" or "production mode" with the flick of a switch (the TEST parameter). When in "test mode" the job will execute additional code and steps that are inserted by the programmer for de-bugging purposes. This additional code can remain in the job after de-bugging is complete, allowing easy testing of the job during future maintenance.

Note that there are two semicolons used in the first statement of the above macro example. The first is used to end the macro statement, while the other is a trailing semicolon that will end the resolved SAS statement.

■ Pay attention to your SASLOG
- The SASLOG should be your best friend. Use it!
- Do not ignore SAS warnings. Understand why they are generated and implications for your processing.

■ Know Costs of SAS
- Don't be afraid to use non-SAS jobs. Forgive this blasphemy, but processing large external files may be best done with non-SAS software.

■ Beware of New SAS Features
- As with any software package, new features should be used with caution. They may be costly to utilize, unstable, or have undetected bugs.

A good example is the new MODIFY statement. This statement is used to update a SAS data set in place, which sounds efficient. However, using this statement incurs a huge CPU increase over the more familiar SET statement.

SAS DATA Step

■ Read only the data that is needed
- Eliminate unneeded observations immediately.
- Input only the fields you need from external files.
- Keep only variables you need from SAS data sets.

■ Know How the DATA Step is Processed
- Realize that the DATA step is a loop that is executed for each observation in a data set.

- Know the difference between executable and non-executable statements. Group each type together in your DATA steps. While this doesn't save resources per se, it shows that you have an understanding of SAS that allows you greater insight into efficient SAS DATA step programming.

■ Consider the SAS Program Data Vector (PDV)
- The PDV is a set of buffers that includes all variables referenced, either explicitly or implicitly, in a DATA step (Henderson, et. al., 1991). It is initialized at the top of the DATA step and then loaded with values for each observation processed.

- Note that since the WHERE= data set option and the WHERE statement will cause only desired observations to be loaded into the SAS Program Data Vector, any FIRST. and LAST. variables are set for only the observations that meet the WHERE condition. A subsetting IF statement loads all observations into the PDV before subsetting. Thus, it would set the FIRST. and LAST. variables according to all observations in a data set.

■ Nested Loops

-When nesting loops, place the loop with the fewest iterations outermost. This can provide excellent CPU savings because SAS is jumping around less.

■ Avoid Using LINK and GOTO Statements

These statements also cause the SAS System to jump around, increasing CPU utilization. Using macros to generate repeated code will save resources in this case.

■ Arrays

It has been suggested (Hardy, 1992) that users avoid using SAS arrays because SAS requires much more CPU to process arrays than if each statement is explicitly coded. This is due to the fact that array elements can be of varying length and array indices are floating point numbers (Squillace, 1988).

Because I am using differing performance statistics to measure efficiency on different platforms, I am of the opinion that array efficiencies differs by platform. Using arrays on the mainframe in Version 6.07 cost 30% to 100% more CPU than explicit coding, while the same code on the PC, in Version 6.04 as well as Version 6.08, cost only between 0% and 10% more in elapsed time.

Thus, while arrays always use more resources, they appear to be less inefficient on the PC platform where efficiency is measured by elapsed time.

Although arrays can be easier to code and read, I try to avoid their use on the mainframe. For larger data sets I often use macros to generate statements that would otherwise be processed using arrays.

■ Know SAS DATA Step Language Statements

- Effectively use FIRST. and LAST. variables that are created by a BY statement.
- The RETAIN statement is also extremely useful. But remember that it will only work with variables that are newly created in the DATA step.
- The SAS Applications Guide is an excellent reference for beginning to appreciate the SAS DATA step programming language.

SAS PROCS

■ Use PROCs for their intended purpose

- PROC SUMMARY uses somewhat less CPU than PROC MEANS because PROC MEANS was designed to generate printed output while SUMMARY does not.

■ Use SAS PROCs to process your data

- Do not waste time writing a DATA step that replicates the actions of a PROC.

- The following PROCs are good data manipulators: APPEND, COMPARE, COPY, DATASETS, EXPAND, SQL, SUMMARY, TRANSPOSE

- Use the CNTLIN= data set with PROC FORMAT.

■ Use output data sets created by SAS PROCs

- Most SAS PROCs produce an output data set that can be creatively used. Some of my favorites are:
  CNTLOUT= data set with PROC FORMAT.
  OUT= data set with PROC CONTENTS.
  OUTEST= data sets on modeling PROCs.

■ Use the NOEQUALS option with PROC SORT

This option specifies that observations with identical BY variables do not need to retain their same relative positions in the output data set as in the input data set.

```
PROC SORT DATA=SAS_DS  NOEQUALS;
```

While the results were uneven on the mainframe Version 6.07, the NOEQUALS case rarely performed worse. On the PC Version 6.08, however, this option usually gave fair elapsed time savings.

■ Use PROC SQL

When processing SAS data sets, PROC SQL can perform multiple operations on multiple SAS data sets that would normally require the use of numerous other SAS DATA steps and procedures. This consolidation of multiple DATA steps and PROCs can be very efficient.

For example, to merge and summarize two data sets would require two PROC SORTs, a DATA step, and a PROC SUMMARY. These actions can all be accomplished with one PROC SQL with as much as a 50% reduction in CPU and elapsed time.

While PROC SQL is good at performing multiple operations on data sets, it is not generally efficient when performing a single function on a single data set.

Unfortunately, PROC SQL requires a fair amount of programmer time to learn due to the unfamiliar syntax (unless you know SQL). However, testing indicates that time spent learning this procedure would be a useful investment.

■ Use the Pass-Through Facility in PROC SQL
- To access a data in a database management system (DBMS) supported by SAS/ACCESS (DB2 table, ORACLE database, ODBC database, etc.), the pass-through facility of PROC SQL can be used. This facility is used to pass logic directly to a target DBMS, allowing the evaluation of a "subquery" to be done by the database system. This is extremely efficient because the data can be queried in its natural state, realizing the inherent efficiencies of the database system (Technical Report P-222).

The following example shows a sample query using the SQL Pass-Through Facility to retrieve data from a DB2 table:

```
PROC SQL;
  /* CONNECT TO connects to DB2 */
  CONNECT TO DB2(SSID=DSN2);
  SELECT *
  /* CONNECTION TO retrieves from DB2 */
  FROM CONNECTION TO DB2
    (
    /* "Subquery" is processed by DB2 */
    /* Passed code is DBMS specific   */
    SELECT *
         /*  DB2 Table  */
        FROM ADT.VMES_IUS_DATA
        WHERE......
    /* End of "Subquery"            */
    /* Results passed back to SAS  */
    );
QUIT;
```

The Pass-Through Facility is one of my most prized SAS features. By being able to pass logic to a target DBMS, I can utilize the efficiencies of the DBMS to obtain only the data of interest from the DBMS. Thus, SAS is passed only the data that is needed.

Unfounded claims of efficiency

The SAS Institute has published a guide of efficient coding techniques called "SAS Programming Tips: A Guide to Efficient SAS Processing". In this guide there are many techniques that are listed, along with the type of efficiencies gained.

Unfortunately, this manual was benchmarked by the SAS Institute using SAS Version 6.04. Beginning with SAS Version 6.07, an optimizing compiler was implemented. This new compile routine appears to have obviated many techniques recommended for CPU savings in this SAS manual. In my testing of these techniques, I have found many of these recommendations to no longer be helpful for CPU savings.

This does not in any way, however, diminish the usefulness of this manual as a source of efficient coding techniques. The techniques described in this manual go into much more detail regarding the specific types of resources that can be saved.

Data Set Compressing

Compressing large data sets saves physical storage space and, therefore, I/O. In a compressed data set, SAS treats each observation as a single string of bytes and compresses multiple occurrences of the same byte without regard for variable boundaries. SAS Version 6 can create these variable-length observations only for direct access data sets (Engines V607 or V608).

Because of this compressing algorithm, data set compressing works best for data sets with a large record length (large # number of variables), especially those with lots of missing and/or repeated values. Space reduction typically ranges from 5% to 35%.

Surprisingly, elapsed time on the PC decreased slightly for most operations when using compressed data sets. This was true whether the data being used was resident on the network server or on the PC's hard drive. Evidently, the PC file structure lends itself rather well to the SAS data set compressing routine. Thus, I habitually compress all data sets on the PC.

On the mainframe, however, a completely different story emerged. CPU usage increased by a substantial amount. Therefore, you should first gather information to aid in determining whether a mainframe data set is best stored as compressed or un-compressed. How often the data set will be processed or updated is a key consideration.

I use the following formula as a first guess to evaluate whether compressing a data set might save money:

(Monthly Space Charge) * (% Storage Reduction)
>
(Monthly CPU Charge to Process the data set) * 3

When this condition is true, the amount saved on the mainframe by compressing will likely be greater than the increase in CPU expense that will be incurred. The factor of 3 is used, because average overall CPU charges usually increase by less than 300% when working with compressed data sets.

Again, this estimate is just a first guess. This is a difficult estimation because the overhead of compressing and de-compressing depends on many factors, including the % reduction in space, the proportion of numeric vs character variables, etc.

To create a compressed data set, use the COMPRESS= data set option when creating the data set. This option only needs to be specified when first creating the data set. For example:

```
DATA SASLIB.NEW_DS(COMPRESS=YES);
                or
PROC SORT       DATA=OLD
                OUT=NEW(COMPRESS=YES);
```

NOTE: Observations in a compressed data set cannot be accessed using the POINT= option or by observation number in SAS/FSP procedures.

Stored Program Facility

Use the Stored Program Facility to store and access machine code generated by a DATA step. Store compiled SAS DATA steps for large, frequently run jobs.

"Stored Program Facility" is a bit of a misnomer since only single DATA steps may be stored. SAS PROCs cannot be stored as compiled code.

This technique saves the most when large DATA steps are stored, especially when they are used to process small data sets. This is because the savings come from avoiding the compile time for the DATA step. This is usually a small portion of a SAS job cost.

To store a DATA step as compiled code, use the PGM= option of the DATA statement to specify the name of a SAS file to contain the compiled code as shown below:

```
/* Compile and store a DATA step  */
DATA NEW / PGM=SASLIB.STORED;
   SET IN_DS;
   KG = LBS * .4536;
   ...many more SAS DATA step statements...
RUN;
```

Once the compiled code is stored, it may be executed by referencing the compiled code via the PGM= options of the DATA statement as in the following example:

```
/*  Execute a Stored Program  */
DATA PGM=SASLIB.STORED;
RUN;
```

The optional REDIRECT statement is available when executing the stored DATA step to change the name of the input or output SAS data sets.

Unfortunately, SAS data set options may not be passed via the REDIRECT statement. Also, the data set specified in a REDIRECT statement must have the same PDV as the original data set that was used to created the stored program.

```
/* Execute the stored DATA step */
/* With REDIRECT statements     */
DATA PGM=SASLIB.STORED;
   REDIRECT OUTPUT NEW = NEWLIB.NEW_DS;
   REDIRECT INPUT IN_DS = NEWLIB.IN_DS;
RUN;
```

For a job with many DATA steps, each DATA step must be stored as a separate compiled member in a SAS library. Thus, early planning is advisable to implement this feature in an application due to the complexity of converting an application that contains many DATA steps.

The DATA step used to test this feature contained 320 SAS DATA step statements and showed that there is very little overhead associated with this feature of the SAS System. Savings will increment with each execution of the compiled code. Use of the Stored Program Facility, then, should be encouraged for the proper situations.

SAS Data Views

Use data views to avoid storing data elements in numerous data sets. A data view does not contain any data values. Instead, it simply contains a compiled DATA step program or SQL query that defines data or describes data that is stored somewhere else. Because of this, data cannot be modified in a view.

Views can be created using a number of different methods. The method used should depend upon the needs, complexity, and uses of the view.

SQL views, while requiring knowledge of PROC SQL, are generally more efficient. This is because of PROC SQL's powerful data manipulation capabilities and the SQL query optimizer, which when passed multiple and/or layered views (views that reference other views), will optimize the entire set of compiled SQL queries together.

```
/*  Create an SQL VIEW  */
PROC SQL;
  CREATE VIEW SASLIB.MALE AS.....;
QUIT;
```

DATA step views are the most flexible because of the availablilty of DATA step programming statements. Usually, though, this type of view is more costly than SQL views.

To create a DATA step view, specify the VIEW= option on the DATA statement. Note that the view name must appear twice in the DATA statement as shown below.

```
/* Create a DATA step VIEW */
/* The VIEW name must appear twice */
DATA SASLIB.MALE/VIEW=SASLIB.MALE;
  SET SOCAL(WHERE=(SEX='M'))
      NORCAL(WHERE=(SEX='M'));
   ...other SAS data step statements....
RUN;
```

DATA step views can also be created for external files and, using the SQL Pass-Through facility, SQL views can be created for SAS/ACCESS supported DBMS databases. Once the view of the external file or DBMS database is created, jobs may reference the view, eliminating the need to know the layout of the external file or DBMS database.

The following DATA step creates a view of an external file. When data is needed from this file in future jobs, it can simply be referenced by the SAS data view name SASLIB.HOSP.

```
DATA SASLIB.HOSP/ VIEW=SASLIB.HOSP;
  INFILE   'DOR.HOSP.YRS7992';
  INPUT @1    ID            $7.
        @8    GENDER    $1.
        @9    DOB       MMDDYY6.
        ..............
        ;
  AGE = (TODAY()-DOB)/365.25;
RUN;
```

See the previous section on how to use the SQL Pass-Through facility for an example of how to reference a DBMS database. As above, simply use the CREATE VIEW statement to create a view of the DMBS database.

Note that SAS/ACCESS can also create SAS data views, but this method will not be discussed in this paper. This type of view is the most costly and least flexible.

Like the Stored Program Facility, code that defines a SAS data view does not execute. It is simply compiled and stored for future use. Once stored, the view is referenced like a SAS data set.

The main advantages of using SAS views include:
- Complex views can be created for easy use by novice programmers or to limit access to data in a master data set.
- Storage space is saved since only the view definition is saved instead of a whole data set.
- The input data is always current because data from views are derived at execution time.

Be warned, though. Each time a view is referenced it must execute the underlying compiled SAS code to create the data before performing any requested actions. So, for example, running a PROC PRINT on a SAS data view is like executing a DATA step (or SQL Query, depending upon the type of view) followed by a PROC PRINT.

Since this overhead is associated with a SAS view each time it is referenced, SAS data views are very costly if frequently used. If a view is going to be referenced more than once during a SAS session, you are well advised to create a SAS data set that contains the data described by the view and process the data set rather than the view.

See Norton (1994) for a more extensive discussion of SAS views.

Notes: Stored Program Facility & SAS Views

1) Be sure to keep a copy of the original, uncompiled, SAS code. Source code cannot be retrieved from the compiled SAS code.

2) A stored DATA step can take from 1.5 to 8 times more disk space than an uncompiled SAS program (Reynolds, 1992).

3) Global statements (TITLE, FOOTNOTE, FILENAME, LIBNAME, OPTIONS) and most host-specific options and host-specific data set options cannot be included in a stored DATA step. Macro code cannot be stored either, since the stored code is compiled SAS code.

Conclusion

When it comes to efficiently coding in SAS, it is important to start with the basics. Although the SAS System has many appealing features, many of these pleasantries come with a price tag in terms of CPU usage. Therefore, you should use these bells & whistles only for specialized circumstances where their use is warranted.

Also, don't be needlessly clever. Ad-hoc programs and queries do not need to be coded to be incredibly efficient. Nor will efficient coding techniques produce discernable differences with small data sets.

But, by making basic efficiency techniques part of your everyday coding you will discover that all of your programming has become more efficient. Mastering the basics will take you a long way towards being an efficient SAS programmer.

Remember that while one technique may be very efficient on one platform there is no guarantee that it will work as well on another platform. As shown, different results are obtained depending upon platform and how efficiency is measured.

On the PC platform, I was pleasantly surprised at the speed of Version 6.08 over the previous PC Version 6.04. However, this increased speed will only materialize on PC's with ample resources (486, math coprocessor, 8 megabytes memory).

Finally, of all the newer SAS capabilities, PROC SQL stands out above the rest. I encourage you to learn this procedure because of its powerful, and often unique, abilities.

References

Creech, Stephanie K., Van Wyk, Jana D. (1991), "Using SAS/CONNECT and SAS/ACCESS Software to Access Data in a Distributed Environment", *Proceedings of the Sixteenth Annual SAS Users Group International Conference*, 572-579

Hardy, Jean E. (1992), "Efficient SAS Software Programming: A Version 6 Update", *Proceedings of the Seventeenth Annual SAS Users Group International Conference*, 941-945

Henderson, D., Rabb, M., and Polzin, J. (1991), "The SAS System Supervisor - A Version 6 Update", *Proceedings of the Sixteenth Annual SAS Users Group International Conference*, 249-257

Norton, Andy (1994), "The View Review: PROC SQL Views, SAS/ACCESS Views, and DATA Step Views", *Proceedings of the Nineteenth Annual SAS Users Group International Conference*

Reynolds, Herb (1992), "A Tool for Compiling Data Steps in Large SAS Systems", *Proceedings of the Seventeenth Annual SAS Users Group International Conference*, 111-116

Squillace, Daniel J. (1988), "Producing Efficient SAS Applications", *Proceedings of the Thirteenth Annual SAS Users Group International Conference*, 82-88

Wilson, Steven A.. (1994), "Developing a SAS System Autocall Macro Library as an Effective Toolkit", *Proceedings of the Second Annual Western Users of SAS Software Conference*

SAS Institute, Inc. (1987), *SAS Applications Guide, 1987 Edition*, Cary,NC: SAS Institute Inc.

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

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

SAS Institute, Inc. (1990), *SAS Language: Reference, Version 6, First Edition*, Cary, NC: SAS Institute Inc.

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

SAS Institute, Inc (1991), *SAS Technical Report P-222 Changes and Enhancements to Base SAS Software*, Cary NC: SAS Institute Inc.

Author's Address

Steve Wilson
Kaiser Permanente, Division of Research
3505 Broadway
Oakland, California 94611
(510) 450-2211

INTERNET: saw@dor.kaiser.org

Any comments, questions, or details of your experiences are most welcome.

SAS, SAS/ACCESS, SAS/CONNECT, and SAS/FSP are registered trademarks of the SAS Institute Inc., Cary, NC, USA

SQL and DB2 are registered trademarks of International Business Machines Corporation.

Oracle is a registered trademark of Oracle Corporation.

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