TECHNIQUES FOR EFFICIENTLY ACCESSING AND MANAGING DATA

Steven A. Wilson, Resource Biometrics, Inc., Emeryville, 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 of basic efficiency techniques are listed, some SAS System features are discussed from an efficiency viewpoint. These features include PROC SQL, SAS data set compressing, the Stored Program Facility, and SAS 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, documentation, 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 (e.g., 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 or least utilized platform, reliability can become a problem. My experience is that the least utilized platform is usually the least reliable in operation.

Note that to fully realize this goal, your site must have SAS/CONNECT and SASACCESS installed and operational. Since this is beyond the scope of this paper, refer to Wilson & Terdiman (1995).

### 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 a graphical format.

These tests used 12 different SAS 6.07 data sets, each containing a mixture of randomly generated discrete and continuous numeric as well as character variables of varying lengths. Tests 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 using SAS 6.08.

In fact, the SAS Institute recommends that the SAS System be run on a PC with at least a 486 and 8 meg of memory.

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 usually billed back 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 - The 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.
  - Functional design specifications, if properly done, can lead directly to good coding specifications.
  - Flowcharting helps......honest!
  - Especially for large systems, the more design you can do prior to writing a single line of code the better.
• 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
  This is an important but often ignored part of the job. Documentation is especially important for large systems and in cases where 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. As a rule, I always assume that thorough documentation will consume approximately 25% of the total time to complete any large system.
- 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. I generally use two types of comment boxes that are easily distinguishable when looking at source code:

```plaintext
/*---------------------------------------*/
/* General source code comments */
/*---------------------------------------*/
```

versus

```plaintext
/*****************************/
/* Special comments regarding future */
/* enhancements or problems. */
/*****************************/
```

- Comments of the form /message/ are compiled, and hence more costly, than comments of the form /message/, which are not compiled. However, this savings is miniscule.

• 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 job 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).

• Use of Control Data and Code Generators
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 data sets and/or formats to control actions to be performed and menu selections available to a user. These types of applications tend to be more flexible because they are controlled by values in a data set or format table.

These controls data sets, once created permit you to easily generate source code according to the contents of the control data set.

• Use SAS/ASSIST
The SAS/ASSIST product is a great tool, both for novice and expert SAS programmers alike. This product allows users to quickly generate SAS code using a point-and-click interface. This generated code is automatically bug-free and expertly coded and may be incorporated into other SAS programs.

With SAS/ASSIST, I may never again have to pull out my SAS manuals to look up the specific syntax for a messy PROC TABULATE or PROC GCHART.

Using SAS Effectively
• 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.
- Do not perform unnecessary sorting.

• 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 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 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.

• Effectively Position SAS Statements
As a rule, you should place data selection and manipulation logic as close to the data source as possible.
• Use SAS System 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 BOFNO=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.

• 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.

• Use Formats and Informats

- SAS formats save storage space.

- Saves CPU when used to perform complex assignments.

- Create your own permanent SAS format catalog.

- As previously mentioned, I frequently use formats to control program actions. Thus, rather than changing source code to modify a program's behavior, I can simply modify a format. For example,

```sas
IF TODAY() - DATE > 5 THEN ....
```

• 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 a minimal amount of CPU.

- Learn the options that can be used on the LIBNAME and FILENAME statements. Most of these options are host specific, but there are many useful options available.

• 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). This is especially true if the data set is not indexed by the WHERE variables.

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 than simply passing the large data set to each PROC FREQ with a WHERE= data set option or WHERE statement.

• 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.

• 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. In fact, I cannot imagine coding in SAS without the use of the macro facility.

- 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.
- Beware that 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”. This method also allows you to execute code in a “test mode” that would not update any permanent data sets. For example:

```sas
%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;
  %else %do;
    PROC COPY IN=WORK OUT=SASLIB;
    RUN;
  %end;
%mend 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 debugging 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.

• 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. These problems will most likely be resolved in later releases, which is why waiting to implement new features is usually a good idea.

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.

• 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 3 bytes (2 bytes is the minimum for MVS machines).

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.

• 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.
- Comparison operators always result in a numeric true/false result. For example, the expression IF X NE ' ' would result in the value 1 (true) if X is not null or the value 0 (false) if X is null. Creatively used, this fact can be quite useful in some circumstances. For instance:

```sas
NUM_CASE=`CASE1 NE ' ')+(CASE2 NE ' ')+(CASE3 NE ' ')+(CASE4 NE ' ');
```

I was rather confused the first time I saw code like this. Knowing that each comparison operation returns a 1 or 0 allows me to “count” the results from multiple comparisons. NUM_CASE, then, could be in the range 0-4, depending upon the number of “true” comparison results.

- The SAS Applications Guide is an excellent reference for beginning to appreciate the SAS DATA step programming language.

• 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 declarative DATA step statements. Declarative statements supply information to SAS and take effect when the DATA step is
compiled, while executable statements result in an action being performed at execution of the DATA step.

Group each type of statements 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.

- Even if you have been using SAS for a few years, it would be useful for you to review chapter 2 of the SAS Language Reference manual that discusses the SAS DATA step.

**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.

Knowing how the PDV is defined and loaded with values will allow you to understand how the SAS DATA step operates and make debugging a DATA step much easier. Again, I refer you to chapter 2 of the SAS Language Reference manual for additional information.

- Note that since the WHERE= data set option and the WHERE statement will cause only desired observations to be loaded into the SAS PDV, any FIRST. and LAST. variables are set for only the observations that meet the WHERE condition. A subsetting IF statement, on the other hand, loads all observations into the PDV before subsetting. In this case the FIRST. and LAST. variables would be set according to all observations in a data set, resulting in the possibility that FIRST. or LAST. condition testing may never be true.

**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 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 differ 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 (impacting the utilization of programmer time), 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.

**SAS PROCS**

- **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. This option allows you to easily create a SAS format from a SAS data file.

- **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 PROC SUMMARY does not.

- **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 PROCESS.
  - 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.

- **Learn Screen Control Language (SCL)**

  - When using SAS/AF or SASIFSP, the use of SCL is unavoidable. The functionality of SCL is truly awesome and it can be a rather daunting task to learn how to effectively use SCL. However, SCL can make you applications much more user friendly.

  - Some functionality in SCL cannot be found anywhere else in the SAS System. You can execute PROC DISPLAY in a non-interactive mode to utilize this functionality in a background SAS session. For example, the following SCL entry could be called in a batch or background job to determine if a data set is currently locked by another user.
INIT:
   DSN = SYMGET('dsn');
   DSID = OPEN(DSN);
   CALL SYMPUT('locked',DSID);
   IF DSID THEN RC = CLOSE(DSID);
RETURN;
MAIN: RETURN; TERM: RETURN;

- 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. Also beware that some SQL code, although only a couple of simple clauses long, may cause SQL to read the data multiple times, greatly increasing resource utilization.

Because of this, as well as the unfamiliar syntax (unless you know SQL), PROC SQL generally requires a fair amount of programmer time to learn. However, my experience indicates that time spent learning this procedure would be a useful investment.

- PROC SQL allows you to access special dictionary tables to retrieve information about all SAS data libraries and external files that are allocated in a SAS session. These tables are automatically maintained by the SAS System and provide you with information on the contents of all allocated SAS files. These special dictionary tables are also maintained as SOL views in the SASHELP library, allowing you to use these dictionary tables in other procedures and DATA steps. (See Technical Report P-222 for further information.)

For example, the following code would list all the variables contained in all the data sets in all currently allocated SAS libraries:

   PROC PRINT DATA=SASHELP.VOCOLUMNS;
   RUN;

- Use the Pass-Through Facility in PROC SQL
  To access 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 *
   FROM CONNECTION TO DB2
   (/ * "Subquery" is processed by DB2 */
   / * Passed code is DBMS specific */
   / * DB2 Table */
   FROM ADT.VM3ES_UUS_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 and processes only the data that is needed.

Unfounded claims of efficiency
As previously mentioned, 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 great detail regarding the specific types of resources that can be saved.

Data Set Compressing
Compressing large data sets saves physical storage space and 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, V608, V609, and V610).

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, CPU usage increased by a substantial amount. Testing, then, is imperative for making a determination as to 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.

The expense of compressing and de-compressing depends on many factors, including the % reduction in space, the proportion of numeric vs character variables, and the efficiency of your host at I/O operations.

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

```sas
DATA SASLIB.NEW_DS(COMPRESS=YES);

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 the machine code that is 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:

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

The above DATA step does not execute. Instead, the code is simply compiled and stored for later execution.

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:

```sas
/* 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 exact same PDV as the original data set that was used to created the stored program.

```sas
/* 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.

**Stored Compiled Macro Facility**

Like the Stored Program Facility, the SAS macro language allows you to store compiled macros. This feature, too, is recommended for often used macros or environments that rely heavily upon autocall macros.

To store a macro as compiled code, the SAS System options MSTORED and SASMSTORE=/libref must be specified. Then submit the macro source code with the special STORE option on the macro statement as shown below:

```sas
OPTIONS MSTORED SASMSTORE=/libref;
%macro storeme / STORE;
    ...macro source code....
%mend storeme;
```

The above macro is simply compiled and stored in the catalog entry /libref.SASMACR.STORME.MACRO. The compile time for this macro is avoided when you can execute the stored compiled macro.
SAS Views

There are many different types of data views in the SAS System. Regardless of the type of SAS view used, the view does not contain any data values. Instead, views contain machine code that defines data or describes data that is stored in another place. Views, then, can be used to avoid storing data elements in numerous data sets.

Below is an outline listing the various types of views available in the SAS System:

View of SAS files
  - SQL view
  - DATA step view

View of non-SAS files
  - SQL Pass-Through view *
  - DATA step view
  - SAS/ACCESS view *
  (* requires necessary SAS/ACCESS interface)

The type of view used should depend upon the needs, complexity, and uses of the view. Since a view does not contain any data, data cannot be modified in a view (except in the special case of SAS/ACCESS views).

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 availability 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'));
  SET 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.

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 DBMS database.

Wilson & Terdiman (1995) contains a detailed comparison of views used to access DBMS tables using SAS/ACCESS. ACCESS views are not discussed in this paper.

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;

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 Regarding Stored Compiled Code

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

2) Stored compiled code can take from 1.5 to 8 times more disk space than uncompiled SAS code (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, except in the case of stored 
compiled macros, 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 platform.

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 meg).

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

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, 1434-1439

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 Nineteenth Annual SAS Users Group International 
Conference, 92-97

Wilson, S., Terdiman, J. (1995), "Interoperability Features of 
the SAS System: Administering and Using SAS/ACCESS 
and SAS/CONNECT", Proceedings of the Twentieth Annual 
SAS Users Group International Conference


SAS Institute, Inc. (1993), SAS Companion for the Microsoft 
SAS Institute Inc.

SAS Institute, Inc. (1990), SAS Companion for the MVS 
Institute Inc.

SAS Institute, Inc. (1990), SAS Language: Reference, 


P-222 Changes and Enhancements to Base SAS Software, 
Cary NC: SAS Institute Inc.

Author's Address
Steve Wilson 
Resource Biometrics, Inc. 
5801 Christie Avenue 
Emeryville, California 94608-1928 
(510) 597-7200 
INTERNET: wilson@resbiom.com

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

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

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

A shorter version of this paper of the same title was 
presented at the 1994 SAS Users Group International, the 
1994 Western Users of SAS Software, and the Wisconsin-
Illinois SAS Users Group conferences.