CHAPTER 1

Introduction

What Is Efficiency?

To be efficient, a program must:

- use the least costly set of resources, and
- accomplish the programming objectives.

Both criteria sound straightforward. Neither one is quite so simple.

Evaluating the Cost of Resources

Consider the first issue, using the least costly set of resources. Each computer system will include a variety of hardware and services. For example, you may pay dearly for:

- disk space
- CPU time
- I/O

- technical support
- tape drives
- turnaround time.

Even if your fee schedule does not list all of these items, you may pay for them indirectly. For example, when using a PC, there is no direct charge for CPU time. However, the longer a program runs, the longer you must sit and wait for it to finish. On a mainframe, you may not pay for using tape drives. However, jobs that use many tape drives may have to wait until the tape drives become available, perhaps running overnight.

An abundant resource at one site might be scarce at another site. At one site, disk space might be plentiful and CPU time costly. At another site, the reverse might be true. One site might provide a Help Desk with a senior programmer who knows the SAS System extremely well. At another site, you might be lucky to get a manual. Under different cost structures, you might use entirely different programming techniques because the relative costs of the resources change.

Consider three ways of storing this entirely fictitious data. Structure #1 is the most flexible for analysis purposes:

```
Variables:          TEAM      WINS      YEAR

 Values:         RED SOX       80        1997
                 RED SOX       80        1998
                 RED SOX       85        1999
                 TIGERS       105        1997
                 TIGERS        90        1998
                 TIGERS        80        1999
                 YANKEES       80        1997
                 YANKEES       90        1998
                 YANKEES      110        1999
```

Structure #2 might work best for printing the data or converting it to spreadsheet form:

Variables:	TEAM	WIN97	WIN98	WIN99
Values:	RED SOX	80	80	85
	TIGERS	105	90	80
	YANKEES	80	90	110

Structure #3 is best for generating plots on an x-y axis:

Variables:	RED_SOX	TIGERS	YANKEES	YEAR
Values:	80	105	80	1997
	80	90	90	1998
	85	80	110	1999

Which structure is best? As you sit down to discuss the matter with your boss, you might say, "It really doesn't matter which form we use. I can write a program to convert from one form to another whenever it becomes necessary." And your boss might reply, "I know you can. But I have 10 other users who can't. They have to do the research, and they need the data available in all three forms. So store it all three ways or else they'll be bothering you for help all the time." And you say, "But, but, but..." And the boss says, "No, no, no." Who's right?

Grudgingly, you may have to admit that the boss might be right. The bottom line is that there is no one right answer. It might be cheapest to store the data three times instead of once, and the best strategy might differ from one project to the next. Besides the dollar cost for storage space, take into consideration

- the skill level of the users
- the amount of free time you have to help out the users

- the number of expected changes to be made to the data in the future
- whether you, as the senior programmer, have the skill to store the data in one form, and store the other forms as views rather than data sets.

Does the Program Work?

What about the second criterion for efficiency: Does the program accomplish its objectives? This too sounds like a clear-cut issue; the program works or it doesn't. Once again, gray areas abound. Consider the possibilities below.

When the program contains a syntax error, it certainly did not accomplish its objectives. However, be sure to check the SAS log. The existence of output does not guarantee an error-free program. Perhaps the first half of the program produced valid output, while a later SAS step contained an error. Also, if your method of running the SAS System produces .LOG and .LST files as output, the existence of a .LST file does not guarantee zero syntax errors. It is quite possible that the latest run of the program contained many errors, and that the .LST file was produced by running an earlier version of the same program.

When the program contains a logic error, the output can be wrong. Sometimes, notes in the SAS log will provide a clue. Messages about numeric to character conversion, lost card, or variables being uninitialized sometimes indicate a logic error. Still, many logic errors, such as faulty IF / THEN / ELSE logic, produce no messages to help you.

A program can be too complex. Have you ever seen a 200 line program with no comment statements, no spacing or indention, and complex programming techniques? It may work perfectly. But nobody could possibly know this because nobody can figure out how the program produces its results.

Perhaps the program works, but it has to run overnight because it ties up key resources such as an important data set or too many tape

drives. The longer you have to wait for the results, the less efficient the program.

Perhaps the program would work, but it is too expensive to run.

Finally, what if the program works, but it takes a lot of effort to interpret the results? A more efficient program would clearly display what the user needs to know.

Weighing the Benefits and Costs

Many considerations go into labeling a program as efficient or inefficient, but they all fall into these two basic categories: the efficient program uses the least costly set of resources, and the program accomplishes the programming objectives. When this book presents a technique that might speed up your program, you still must decide whether it makes sense for you to apply the technique. When evaluating a technique, consider these issues:

- How much does this programming technique save? If your program processes a small data set, you shouldn't spend three hours of your time to cut a program's CPU time in half. Half of nothing is nothing.

- How often does the program run? If you save 5% of the CPU time for a program which runs daily, that can add up over the course of a year.

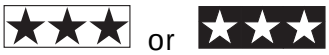
- Do you feel comfortable with the programming technique? Practice makes perfect. But if you are uncertain whether the results are accurate, the program is useless.

- Do you have the time to spend investigating alternative programming techniques? Perhaps your workload is too heavy to worry about trying something new. Perhaps the program results are needed quickly.

My bottom line recommendation is this: learn some new techniques for improved efficiency and make them habits. Over time, incorporate more of these into your programs.

Rating the Savings

To help you evaluate the importance of various sections, this book rates the topics and programming techniques, using one to three stars:

 or

Saves the most CPU time or storage space

 or 

Medium savings

 or 

Smallest savings

Black stars vs. white stars reveal my assessment of whether the technique applies frequently (black stars) or infrequently (white stars). Naturally, you may find that frequency in your programs differs from my assessment.

Two additional icons cover special situations:

The savings fluctuate widely, depending upon your operating system or the characteristics of your data. The savings can range from large to nonexistent to negative.

This technique does not save. It may have, under previous releases of the software.

If you notice that a technique would be useful in many of your programs, add another star. These ratings are general guidelines only, to help you prioritize which material is most valuable to learn first.

Be careful when interpreting the percentage savings offered by alternative statements (discussed in Chapter 10). The percentage savings in CPU time represents the savings for that one statement, not the savings for the entire DATA step. Also, note that saving 10% of the CPU time when the bill is $200 is more valuable than saving 25% when the bill is $10.

The Focus of This Book

This book focuses on getting programs to run faster. Most of the book addresses reducing CPU time (Chapters 2 through 7), with one chapter on saving storage space (Chapter 8). Often, I made an executive decision to place a topic in one chapter or another. For example, the CLASS statement in PROC MEANS works on unsorted data. By replacing a BY statement, you eliminate the need for PROC SORT. I placed this topic in Chapter 5 (sorting) not in Chapter 6 (summarizing data).

For the most part, the book does not distinguish between saving CPU time vs. saving on I/O. Chapter 9 addresses I/O briefly, as well as other general efficiency items that do not fall neatly into earlier chapters. Still, not even Chapter 9 delves into system options that affect memory management or special situations such as extremely large data sets. The intent is to cover programming tools and techniques that come into play for the vast majority of applications programmers.

Finally, Chapter 10 displays the savings generated by the test programs used in writing this book. You can download copies of these programs to verify that the savings on your hardware and the release of the software match the results in this book. This chapter also explains how to generate test data and write programs to test your ideas on efficiency.