Efficient Programming with the SAS® System for Personal Computers: Save Time, Save Space

Lisa Horwitz, SAS Institute Inc., Cary, NC
Tim Thompson, SAS Institute Inc., Cary, NC

ABSTRACT

Users of the SAS® System for Personal Computers are confronted with lengthy processing time and storage limitations not usually encountered when running the SAS System under other operating environments. Therefore, efficient programming, while certainly an ideal in the mainframe and minicomputer worlds, becomes a necessity in the microcomputer world. On the other hand, users of the SAS System for PCs may have greater flexibility than their mainframe and minicomputer counterparts in the method they use to run their SAS jobs and in how they configure their PC environment. Utilizing common sense programming techniques, as well as Release 6.03 syntax and configuration specifications, improves the efficiency, reduces the processing time of SAS programs, and cuts down on storage space for output SAS data sets.

INTRODUCTION

The data file used in this paper consists of information on five thousand households in an imaginary town. A questionnaire (see **Appendix 1**) was created that consists of forty-six questions that might typically be asked in a demographic survey. The type of information requested included the number of adults and children in the household, total income, whether certain appliances were owned, favorite leisure activities, whether the respondent voted in the last presidential election, and so on. The data were generated by a SAS program that took into account the correlation between certain responses, a normal distribution of responses for certain questions, and a reasonable percentage of positive and negative responses for other questions.

The intent of the study was to create a moderately large data file of five thousand records and forty-six fields. With that data file, various programming methods and system configurations were tested. The test goals were to improve performance, increase the speed of analyzing the data, and to explore methods of storing the data more efficiently.

The findings, including the percentage of improved performance in the case of alternative methods of accomplishing the same task, are presented in this paper. These percentages are offered in parentheses, followed by letters indicating the computer used to perform the tests: HP for the Hewlett-Packard Vectra® MEM for Memorex, PCLIM for PC's Limited, and PS for IBM® Corporation's PS/2. For more information on the computers used in these tests, see **Appendix 2**.

PROGRAMMING CONSIDERATIONS

Creating the Data Set

The data from the questionnaires were entered into a standard ASCII text file, PCSURVEY.DAT. The first task was to create a SAS data set from the raw data:

```
libname perm 'c:\sugi';
data perm.survey;
   infile 'c:\sugi\pcsurvey.dat';
   input @1 id 4.          @5 address $14.   @19 phonenum $8.
         @27 adults 1.     @28 children 1.   @29 pets 1.
         @30 income 6.     @36 housing 1.    @37 hometype 1.
         @38 rooms 2.      @40 baths 1.      @41 fire $1.
         @42 garage $1.    @43 cars 1.       @44 stereo $1.
         @45 tv $1.        @46 vcr $1.       @47 cable $1.
         @48 freezer $1.   @49 microwv $1.   @50 dishwash $1.
         @51 washer $1.    @52 dryer $1.     @53 pool $1.
         @54 leisure1 2.   @56 leisure2 2.   @58 leisure3 2.
         @60 grade $1.     @62 fastfood $1.  @63 sitdown $1.
         @64 creditcd 2.   @66 grocery 4.    @70 meat 3.
         @73 dairy 3.      @76 fr_veg 3.     @79 luxury 3.
         @82 bev1 1.       @83 bev2 1.       @84 bev3 1.
         @85 travelfo $1.  @86 travelus $1.  @87 charit 4.
         @91 religion 1.   @92 politic 1.    @93 vote $1.
         @94 radio 1.;
run;
```

Because all of the numeric values in the raw data file were integers, their storage could be decreased from the default size of 8 bytes with no loss of precision:

```
data perm.surveyst;
   infile 'c:\sugi\pcsurvey.dat';
   length id adults children pets housing hometype rooms baths
        cars leisure1 leisure2 leisure3 creditcd bev1 bev2 bev3
        religion politic radio 3;
   length income grocery meat dairy fr_veg luxury charit 4;
   input ...;
run;
```

Three bytes of storage represent all integers through 2^{13}; 4 bytes can represent all integers through 2^{21}. The addition of the LENGTH statement had virtually no impact on the time it took to create the permanently stored SAS data set. However, the size of the data set shrunk from 1,265,092 bytes to 650,092 bytes when the LENGTH statement was used, a size reduction of nearly fifty percent. Although the nature of your data can affect how dramatic the saving of space may be, the inclusion of the LENGTH statement is definitely worth considering.

Sorting the Data Set

Experiments with sorting the data sets were performed next. The larger data set PERM.SURVEY was sorted on one field,

```
proc sort data=perm.survey out=perm.sortlong;
   by grade;
run;
```

and on five fields:

```
proc sort data=perm.survey out=perm.sortlng2;
   by grade politic vote travelfo income;
run;
```

The smaller data set, PERM.SURVEYST, was sorted in the same two ways, first on one field,

```
proc sort data=perm.surveyst out=perm.sortsurv;
   by grade;
run;
```

and then on five fields:

```
proc sort data=perm.surveyst out=perm.sortsrv2;
   by grade politic vote travelfo income;
run;
```

In most of the tests it took slightly longer to sort on five fields than on one, as might be expected. Far more significant was the difference in time it took to sort the smaller data set rather than the larger one. With one BY variable, sorting occurred faster (44% HP, 39% MEM, 46% PCLIM, 47% PS) on the small data set PERM.SURVEYST than on the larger. With five BY variables, the results were in most cases even greater (47% HP, 40% MEM, 45% PCLIM, 53% PS). Clearly, the smaller the data set, the faster the processing. Using the smaller data set also saved on the available amount of work space needed for the sort.

Subsetting the Data

The amount of processing time needed for subsetting a raw data file and a SAS data set with the subsetting IF statement, was compared to subsetting a SAS data set with the WHERE statement, a feature in Release 6.03.

This DATA step reads the raw data and writes the observations to the data set SUBSET1 if income is greater than $60,000:

```
data subset1;
   infile 'c:\sugi\pcsurvey.dat';
   length ...;
   length ...;
   input ...;
   if income>60000;
run;
```

This DATA step accomplishes the same task but reads the data set PERM.SURVEYST:

```
data subset2;
   set perm.surveyst;
   if income>60000;
run;
```

The process of subsetting the SAS data set was much faster, taking less time (79% HP, 80% MEM, 80% PCLIM, 82% PS) to complete.

If the time it takes to create a data set is added to the time needed for subsetting that data set, it is faster to subset from the raw data. For the greatest efficiency, minimize the number of times your applications read data, whether the data are in raw or SAS data set form. Consider how often you want to analyze a particular file of data; the more often you access it, the more it makes sense to store it in the form of a SAS data set because processing a SAS data set can be so much faster.

Another way to perform the same task is to use the WHERE statement. The syntax of the WHERE statement is

WHERE *whereexpression*;

where *whereexpression* is an arithmetic or logical expression, as described in SAS Technical Report P-171, *Changes and Enhancements to Base SAS Software for Personal Computers, Release 6.03*. The WHERE statement is not executable and works before observations are brought into the program data vector; it cannot be used with raw data records. On the other hand, the subsetting IF statement is executable and works on observations that are already in the DATA step. Also, the statement can be used when reading raw data.

```
data subset3;
   set perm.surveyst;
   where income>60000;
run;
```

The WHERE statement proved to be considerably more efficient than the subsetting IF statement; it improved processing time by (15% HP, 28% MEM, 29% PCLIM, 37% PS). The data sets SUBSET2 and SUBSET3 each contained 1290 observations; in such situations, the WHERE statement can be more efficient than the subsetting IF statement because the observations do not have to be written to the program data vector for selection.

In further testing with more complex selection expressions, the WHERE statement consistently ran faster than the subsetting IF statement. The fewer the observations that met the conditions, the greater the improvement in performance with the WHERE statement. For example, in a situation where 269 observations met the two conditions imposed, the DATA step with the WHERE statement ran in less time (43% HP, 53% MEM, 51% PCLIM, 57% PS) than the one with the subsetting IF statement; in a situation where 62 observations met the three conditions imposed, the DATA step with the WHERE statement ran in less time (50% HP, 61% MEM, 56% PCLIM, 63% PS) than the one with the subsetting IF statement.

When the proportion of observations that are written to the output data set is large, the difference in efficiency between the two statements is less significant. In one test, the subset of the data contained 4,888 observations, almost all of the original file. In this case, the processing time for the WHERE statement was still faster than with the subsetting IF statement, but the difference in time was only (5% HP, 6% MEM, 2% PCLIM, 13% PS).

Note that the two statements are not always interchangeable; refer to SAS Technical Report P-171 for several examples of where the statements produce different results.

Creating New Values

Another area of comparison involved the creation of variables. The simplest syntax consisted of numerous IF-THEN statements defining values for a new variable, CATEGORY:

```
data new_var1;
   length category $ 30;
   set perm.surveyst;
   if income<10000 then category='Less than $10,000';
   if 10000<=income<15000 then category='Between $10,000 and
      $15,000';
   if 15000<=income<20000 then category='Between $15,000 and
      $20,000';
         .
         .
         .
   if income>130000 then category='Greater than $130,000';
run;
```

This is not efficient coding. If income is less than 10,000, then the value for CATEGORY is determined in the first test, but every other test in the DATA step is performed anyway.

The following is a more efficient version, running (32% HP, 25% MEM, 30% PCLIM, 29% PS) faster:

```
data new_var2;
   length category $ 30;
   set perm.surveyst;
   if income<10000 then category='Less than $10,000';
   else if 10000<=income<15000 then category='Between $10,000 and
        $20,000';
          .
          .
          .
   else if income>130000 then category='Greater than $130,000';
run;
```

Another way to accomplish the same task is to use the SELECT statement:

```
data new_var3;
   length category $ 30;
   select;
      when (income<10000) category='Less than $10,000';
      when (10000<=income<15000) category='Between $10,000 and
           $20,000';
```

```
      .
      .
      .
    otherwise category='Greater than $130,000';
  end;
run;
```

This version of the DATA step ran in nearly the same time as the version with IF-THEN/ELSE statements. The greatest advantage of the SELECT statement is that it allows for clean, easy-to-read code.

Still another technique for accomplishing the same task is to use the FORMAT procedure to set up a look-up table and then to use the PUT function to assign the values in the DATA step:

```
proc format;
   value inclevel   low - < 10000='Less than $10,000'
                    10000 - < 20000='Between $10,000 and $20,000'
                       .
                       .
                       .
                     13000 - high='Greater than $130,000';
data new_var4;
   set perm.surveyst;
   category=put(income,inclevel.);
run;
```

The DATA step with IF-THEN/ELSE statements ran faster (16% HP, 10% MEM, 15% PCLIM, 6% PS) than the combined time of PROC FORMAT and the DATA step with an assignment statement using the PUT function. DATA steps with IF-THEN/ELSE or SELECT statements ran faster (13% HP, 7% MEM, 12% PCLIM, 3% PS) than the DATA step with the assignment statements, without adding the time for PROC FORMAT. In other words, PROC FORMAT added only about three percent to the total run time.

In another test where five new variables were created, the DATA step with IF-THEN/ELSE logic (creating the data set NEW_VAR5) ran faster (24% HP, 20% MEM, 18% PCLIM, 20% PS) than using PROC FORMAT with five value statements and a DATA step with five assignment statements using the PUT function (creating data set NEW_VAR6). The DATA step with IF-THEN/ELSE logic also ran faster (18% HP, 16% MEM, 15% PCLIM, 16% PS) than the DATA step with the assignment statements using the PUT function alone.

However, if you use PROC FORMAT to set up the values in a table, the formats can be permanently stored and used with other data. In addition, the use of the PUT statements in the DATA step, especially when creating many variables, makes for much more orderly and concise code than large numbers of IF-THEN/ELSE statements.

KEEP= Option

As mentioned earlier in the discussion of sorting the data set, the smaller the data set, the faster the processing. One way to reduce the size of the data set (besides selecting observations) is to select variables; that is, work with only the variables necessary for the application. The processing efficiency of the DATA steps discussed earlier was improved by adding the KEEP= option (the DROP= option could have been used instead). The KEEP= option was added to the DATA statement first, thus controlling which variables were written to the output data set. Again, one new variable, CATEGORY, was created in the DATA step through the use of IF-THEN/ELSE logic:

```
data keep1 (keep=adults children pets income category);
   length category $ 30;
   set perm.surveyst;
   if income<10000 then category='Less than $10,000';
   else if 10000<=income<15000 then category='Between $10,000 and
      $20,000';
      .
      .
      .
run;
```

The data set KEEP1 was created in less time (37% HP, 17% MEM, 12% PCLIM, 25% PS) than the data set NEW_VAR2, which was created without the KEEP= option. Also, because the data set KEEP1 contained only five variables rather than forty-seven, the storage size difference was enormous.

The KEEP= option placed in the SET statement proved to be even more efficient; it controls which variables are read into the program data vector.

```
data keep2;
   length category $ 30;
   set perm.surveyst (keep=adults children pets income);
   if income<10000 then category='Less than $10,000';
   else if 10000<=income<15000 then category='Between $10,000 and
      $20,000';
      .
      .
      .
run;
```

Data set KEEP2 was created in less time (14% HP, 19% MEM, 12% PCLIM, 16% PS) than KEEP1, and in less time (46% HP, 32% MEM, 23% PCLIM, 37% PS) than NEW_VAR2.

These tests were repeated with IF-THEN/ELSE statements to create two variables based on ranges and three variables based on categorical values. Again, adding the KEEP= option to the DATA statement improved performance, but adding the KEEP= option to the SET statement improved performance even more. Using the KEEP= option in both the DATA statement and the SET statement in the same DATA step did not prove to be any more efficient than using the KEEP= option in the SET statement alone, although this was more efficient than using it alone in the DATA statement.

CLASS Statement Versus BY Statement in the MEANS Procedure

Much can be said about the relative efficiency of various procedures based on the type of analysis performed and the type of data involved; only one experiment was done for the purposes of this paper. If the SAS data set you are analyzing with the MEANS procedure is already in sorted form, use the BY statement to get subreports quickly. In fact, comparisons of the time needed to perform PROC MEANS with a BY statement and PROC MEANS with a CLASS statement reveal that the BY statement runs faster (14% HP, 14% MEM, 16% PCLIM, 11% PS).

```
proc means data=perm.sortsurv;
   by grade;
   var income grocery luxury;
run;
```

If the data set is not already sorted, it is much more efficient to use the CLASS statement rather than sorting and using the BY statement:

```
proc means data=perm.surveyst;
   class grade;
   var income grocery luxury;
run;
```

When the time to sort the data set is added to the time it takes to run PROC MEANS with the BY statement, the MEANS step with the CLASS statement proves to be more efficient (68% HP, 74% MEM, 73% PCLIM, 70% PS). The CLASS statement clusters values and makes sorting unnecessary. The format of the report produced with the BY statement is different from the format of the report produced with the CLASS statement.

Programming Suggestions and Summary

Be aware that the size of your data files, the type of processing you are performing on them, the proportion of character and numeric variables, and many other factors affect the overall performance of your SAS programs. The following are general suggestions covered more fully in earlier sections of this paper:

- Minimize the number of times you read your data; especially minimize the number of times you re-create a SAS data set from a raw data file.
- Consider the use of the LENGTH statement to decrease the size of the data set and to speed processing time. Possible truncation problems may occur. Consult the *SAS Language Guide, Release 6.03 Edition*.
- Minimize the number of sorts. If sorting is necessary, perform it on the smallest data set possible. If the goal is to get subreports from PROC MEANS, use the CLASS statement without sorting the data set.
- Consider the use of IF-THEN/ELSE or SELECT statements, or PROC FORMAT and the PUT function for creating new values.
- Make use of the KEEP= or DROP= options whenever possible, especially in the SET statement.

Above all, be familiar with your data and exercise common sense in its analysis. Think small. The sooner observations are excluded from consideration and the fewer the variables carried along through the stages of processing, the faster the processing will be.

PROGRAM EXECUTION, SYSTEM CONFIGURATION, AND MEMORY MANAGEMENT

You can enhance the performance of the SAS System for PCs through your method of running SAS programs, system configuration, and memory management.

Modes of Running SAS Programs

Three modes are available for running SAS programs: interactive display manager mode, interactive line mode, and noninteractive mode.

Interactive display manager mode is a windowing facility for editing and submitting SAS programs, as well as for viewing log messages and output. The SAS Display Manager System also consists of a group of accessory windows for managing a SAS session.

One point that many users of the SAS System for PCs tend to overlook is that temporary SAS data sets exist for the duration of a display manager session. What this means for the user (especially for users from an MVS/OS batch environment) is that once a DATA step is submitted to create a SAS data set, the data set is available any time during the SAS session. Users often submit a DATA step over and over to build a temporary SAS data set when all they need to do is access the existing SAS data set with a procedure.

Interactive line mode also gives you an interactive SAS environment, but you do not have a windowing facility. Instead, you get a question mark for a prompt, and you enter SAS statements. When you press the ENTER key, the statement you entered is compiled immediately; any notes or error messages are displayed at that point. The current step is executed when a RUN, DATA, or PROC statement is encountered. To invoke the SAS System in line mode, issue the SAS command with the -NODMS option:

```
C>sas -nodms
```

Why would you use interactive line mode? If you are using display manager to run SAS programs that involve intensive processing (for example, statistics, complex graphs, or large data files), you may receive out-of-memory messages. If you run into this problem, try running the same program in a line-mode session. Line mode uses approximately 100K less memory.

Noninteractive mode is the final mode for running SAS programs. Like line mode, this mode uses less memory than display manager mode. A big advantage of noninteractive mode is the ability to run SAS/FSP® software applications. This is especially useful for invoking end-user applications; the user can go directly to a SAS/FSP screen without having to deal with display manager. To invoke the SAS System in noninteractive mode, issue the SAS command followed by the name of the DOS file name containing your SAS program:

```
C>sas survey.sas
```

Running Faster

Speed can be an issue when using the SAS System for PCs, especially if you are processing large quantities of data. Complex statistical and graphics procedures tend to increase processing time. You can configure your SAS session to enhance processing speed. You also have some control over configuration of your DOS environment.

The CONFIG.SAS file is a configuration file that the SAS System must find at invocation. This file contains configuration options that identify the directories where executable modules of the SAS System are located. This file can also contain some options that enhance performance.

The -FILEBUFFERS option allows more efficient use of DOS buffers for reading and writing data. The increase in speed is due to read/write operations being performed on groups of records rather than on one record at a time.

The recommended setting for -FILEBUFFERS is

```
-FILEBUFFERS 10 512
```

which specifies ten buffers, each with a length of 512K. The SAS System uses one buffer for each file that is open. If you were writing to a SAS data set, the SAS System would write as many observations as possible to the buffer and then send these data to DOS for writing to the disk. The option works best with data files containing short record lengths.

The -FILECACHE option is another configuration option you can use for improving file access. This option can be used to define a filecache table for any DOS directory. When the SAS System opens a file in a cached directory, it lists the file name in a filecache table. At the point in your SAS program when the file would be closed, it is logically closed by the SAS System instead of physically closed by DOS. When the same file is needed in another program, the SAS System is able to open it more quickly. This option is most effective with executable modules. If you use this option, the number of table entries for all directories should not exceed the value of the FILES= parameter in the DOS configuration file named CONFIG.SYS.

Configuring the DOS Environment If you want to enhance the performance of the SAS System for PCs, an expanded memory board can provide additional flexibility. You can use memory on

an EMS board for disk caching, as a RAM disk, or as a way for the SAS System to utilize expanded memory.

Disk caching is a strategy in which a section of memory is allocated to hold information recently read from disk. Before reading from disk, DOS checks the disk cache area in memory to see if the needed information is already cached. If the information is cached, DOS reads it from the cache area instead of the disk. The disk caching scheme is totally transparent to the user and the software application. Disk caching can greatly increase the speed of SAS processing.

A RAM disk can also increase the speed of software applications. It is possible to assign part of the memory on an EMS board to act exactly like a disk drive. The only difference is that the drive exists only while the PC is turned on. Within the SAS System, the best use of a RAM disk is for the SAS work library (SASWORK directory) and SAS System message files (SASMSG directory). Because disk access involves temporary work files, a RAM disk is much faster than a standard drive. In order for the SAS System to utilize a RAM disk, the CONFIG.SAS file must be modified.

Solutions to PC Memory Problems

The SAS System for PCs is a software product comparable to the SAS System for mainframe and minicomputers. Obviously, a PC does not have the computing power or the memory resources of these larger machines. The biggest problem with the SAS System for PCs is trying to run large, complex programs on large amounts of data. Since DOS cannot access more than 640K RAM, a product as powerful as the SAS System may not have enough memory to complete a complicated program.

One possible solution to a PC memory problem would be to use the micro-to-host link and remote submit your programs to the host. If that is not feasible, your next alternative is to consider the way you are using the SAS System for PCs. For example, suppose you are using display manager to run a SAS program that uses PROC GLM and PROC G3D on a large SAS data set. You get the following message in the log window:

```
ERROR: Cannot load SASG3D  due to insufficient memory.
```

You can try the following solutions:

1. Check to see if any accessory windows of display manager are open. Each open accessory window takes up approximately 24K; make sure you close them when you are finished.

2. Run your program in line mode or noninteractive mode. Running SAS programs without display manager saves approximately 100K.

3. Remove any memory resident programs. This includes pop-up desk managers and terminal emulators.

If you still have memory problems with your applications, the next step is to use an EMS board. The SAS System takes advantage of expanded memory by loading display manager and parts of the supervisor in this area. It frees up approximately 250K in conventional RAM. To utilize expanded memory, you need to add the -EMS option to your CONFIG.SAS file. This option specifies the amount of expanded memory to assign to your SAS session.

Program Execution, System Configuration, and Memory Management Suggestions and Summary

The following is a list of things to consider when running SAS programs. (These topics are covered more fully in earlier sections of this paper.)

- Consider the best way to run your particular applications: interactively with display manager, interactively with line mode, or noninteractively.
- Optimize the -FILEBUFFERS and -FILECACHE options in the CONFIG.SAS file.
- Consider the addition of an EMS board.
- Consider various memory management issues.

Appendix 1. Survey

ID ____ Address _____ Phone No. _____

1. Number of adults living at this address. __
2. Number of children living at this address. __
3. Number of pets (dogs/cats) at this address. __
4. What is the approximate yearly income for the entire household? $ _____
5. What is the approximate number of credit cards owned by your household? __
6. How much in dollars did your household contribute to charities? $ ____
7. Check the highest education level obtained by any member of your household.

   __ High School
   __ College
   __ Graduate School

8. How many automobiles are owned by your household? __
9. Do yon RENT or OWN your current residence? _____
10. Please indicate your type of housing

    __ Apartment
    __ Townhouse
    __ Condominium
    __ House

11. How mauy rooms are in your residence? __
    (do not include bathrooms)
12. How many bathrooms are in your residence? __
13. Check the following items currently in your household.

__ Fireplace	__ Washer
__ Garage	__ Dryer
__ Pool	__ Freezer
__ TV	__ Microwave
__ Cable TV	__ Dishwasher
__ VCR	
__ Stereo	

14. Please choose three (3) of the following leisure activities that your household members enjoy the most.

_ Watching TV
_ Attending/Playing Sports
_ Dining Out
_ Playing/Listening to Music
_ Going to Movies/Theatre
_ Reading
_ Entertaining
_ Shopping
_ Gardening
_ Camping

15. Did you eat at a fast food restaurant in the past 30 days? (Y or N) __

16. Did you eat at a cafeteria or full service restaurant in the past 30 days? (Y or N) __

17. What is your household's weekly expense for groceries in dollars? $ ____

18. Based on the amount you spend on groceries each week, approximately how much does your household spend on the following categories of food?

$ ____ Dairy Products (Milk, Cheese, Buttter, etc.)

$ ____ Fruits and Vegetables

$ ____ Meat, Poultry, Fish, Pork

$ ____ Luxury Items (Candy, Cakes, Pastries, etc.)

19. Based on weekly consumption, choose the three (3) most popular beverages in your household.

__ Soft Drinks
__ Coffee
__ Tea
__ Milk
__ Mineral Water
__ Beer
__ Wine
__ Other

20. What is your religious affiliation?

__ Protestant
__ Catholic
__ Jewish
__ Other

21. What is your political affiliation?

__ Democrat
__ Republican
__ Independent
__ Socialist
__ Other

22. Did you vote in the last presidential election? (Y or N) __

23. What type of radio station are you most likely to listen to? (Y or N)

__ Rock
__ Country
__ News
__ Talk Show
__ Classical
__ Other

24. Have you traveled in the U.S. for a vacation in the past year? (Y or N) __

25. Have yoo traveled outside the U.S for a vacation in the past year? (Y or N) __

Appendix 2. Summary of Programming Considerations

Tests were performed on the following personal computers:

- Hewlett-Packard Vectra®: running at 8 megahertz, 640K conventional memory, 30-megabyte hard disk
- Memorex: running at 10 megahertz, 640K conventional memory with 384K disk caching, 44-megabyte hard disk
- PC's Limited: running at 12 megahertz, 640K conventional memory with 384K disk caching, 40-megabyte hard disk
- IBM® Corporation's PS/2: running at 10 megahertz, 640K conventional memory with 384K disk caching, 44-megabyte hard disk

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

IBM is a registered trademark of International Business Machines Corporation, Armonk, NY, USA.

Vectra is a U.S. registered trademark of Hewlett-Packard Company.