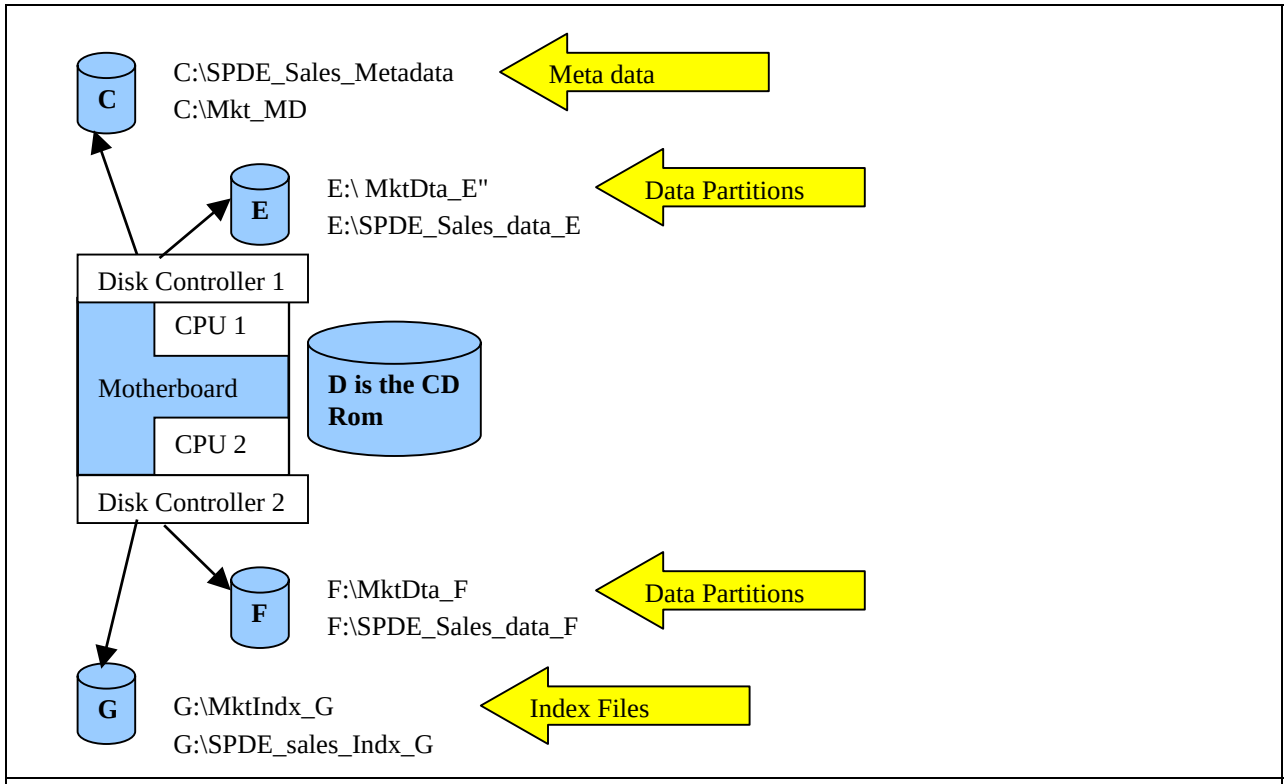
An Annotated Guide: The New 9.1, Free & Fast SPDE Data Engine

Russ Lavery, Ardmore PA, Independent Contractor
Ian Whitlock, Kennett Square PA

ABSTRACT

SAS® has been working hard to decrease "clock time to solution" and provide faster response to users of SAS software. SAS is not relying on computer manufacturers producing faster chips to reduce run times, since faster chips will not give the performance gains you can get with **parallel operations**, especially for I/O. In V9.1, SAS modified the Scalable Performance Data Server (a "For-Fee" product) and created a "free-with-base V9.1 SAS" product called the **S**calable **P**erformance **D**ata **E**ngine (the **SPDE** is often referred to as the speedy engine). With proper equipment, setup, and data, the SPDE engine can cut the I/O time of a SAS job significantly. The amount of the time decrease depends on equipment, the task performed and the data. SAS found that making major reductions in process time required making major changes to the structure of SAS data files. This free data engine has a new file structure, is multithreaded and is accessed through a SAS libname statement. Old file structures will be supported. SPDE is an addition to SAS and none of the old SAS functionality will be lost.

Figure 1 shows two possible SPDE file structures on a 2 CPU-four drive PC. To take advantage of the SPDE engine one must spread file components over the system, giving each component a separate I/O path. Four CPUs would give significantly better performance than two, since there is a fair amount of threading overhead. It seems that on the machine below, the second CPU spends a lot of its time on "threading management" rather than processing the data. Two CPUs are shown because the code was run on a system with only two CPUs.



Sample SPDE libname statements

```
Libname SPSales
    SPDE "C:\SPDE_Sales_Metadata"
Datapath=("E:\SPDE_Sales_data_E"
          "F:\SPDE_Sales_data_F" )

indexpath=("G:\SPDE_sales_Indx_G")
partsize=64;
run;
```

Note Partition sizes

```
Libname SPMktng
          SPDE "C:\Mkt_MD"
  datapath=("E:\MktDta_E"
            "F:\MktDta_F")

  indexpath=("G:\MktIndx_G")
  partsize=128;
    run;
```

FIGURE 1

INTRODUCTION, USER BENEFITS AND USER COSTS

SPDE has a new file structure that is designed to support parallel I/O access (i.e. minimize I/O conflicts). SPDE is designed to improve performance on computers with multiple CPUs, and multiple drives, working on large files. SPDE has a new file structure that differs greatly from the structure of older SAS files.

However, SPDE can also improve performance on a single CPU machine, but not to the extent it can on a multi-CPU machine. Implicit sorting, and index optimization of a WHERE clause with OR conditions, are a couple of examples where SPDE will help with one CPU. In addition, multiple threads can provide some parallelism by allowing one thread to process instructions (as in sorting) while another thread does I/O. If the single CPU machine has multiple data paths, i.e. multiple disk controllers and disks, available, then some performance improvement can still be realized from using multiple threads which can spread the I/O across multiple data paths. Said another way, data can be simultaneously retrieved from multiple I/O devices, ensuring that the processor never has to wait for data; hence insuring that the CPU is fully utilized.

For programmers working on Multi-CPU, Multi-Drive computers and accessing large files, SPDE is fast for several reasons.

- SPDE is multi-threading and, with proper hardware, can perform several tasks at the same time.
- Metadata (indexes, and variable descriptions) are split and separated from data.
- Data can be split into several independently accessible partitions spread over several disks.
- If a dataset is spread over multiple drives, SPDE can perform read/write operations on all drives at the same time.
- If a data step has a where clause, the where clause can be applied, simultaneously, to all data streams.
- Both the SPDE and BASE engines “apply” essentially the same WHERE clauses, but the optimization for the SPDE engine is more sophisticated and therefore optimizes more types of WHERE clauses than the BASE engine can.
- SPDE has what is called “implicit sorting capability”, the ability to deliver sorted data to a By statement, without using a Proc Sort.
- SPDE has a new type of two-part index that is faster that old SAS index.
- SPDE can create indexes in parallel (two, or more indexes at the same time).
- Finally, SPDE can handle very large files and deliver a sorted data stream from an unsorted data set.

The SPDE benefits seen by the user are:

1) shorter clock time to solution, and
2) better use of appropriate (multiple CPUs and I/O paths) hardware.

The user costs are not unreasonable. These costs are: buying more complex hardware, a need for skill enhancement and the problem of dealing with the additional program complexity. Since SPDE is multithreading, it must run on an OS that supports multithreading (Unix, Linux and Windows XP-Pro for example).

SPDE It wants a multi-CPU machine, like a server or a dual-core PC. There is little information available on how to buy a good SPDE desktop, but the white papers suggest: Two CPU’s minimum (and four is preferred) and the computer should have at least one separate I/O path to the disks for each CPU.

The programming task will become more complex because the programmer will have more choices to sort through as s/he programs. Programmers will have to learn the SPDE LIBNAME statement syntax in order to create SPDE files. When dealing with both version 6 and version 7 data one had to deal with multiple LIBNAME statements where one might have been used before. You will be able to do the same sort of thing when working with SPDE data sets and BASE data sets. For example you can merge SPDE and Base in one data step (example shown below) and Proc Copy from one type to another.

The main increase in programming effort in using SPDE will occur when tuning for optimal performance. That task requires considering buffer sizes, block sizes, schemes for partitioning the dataset into its partitions, creation/maintenance of indexes and available disk space. As an example of program complexity, imagine we want to create 5 indexes on a SPDE table. The SPDE can create all five indexes in one pass thorough the file being indexed. However index creation involves the creation of temp files for each index. It is possible that a system has enough disk space to hold 5 created indexes but not enough disc space to hold the temp files associated with creation of 5 indexes at once. In that case, the programmer might create three indexes in one operation and the final two indexes in another operation.

UNDERLYING CONCEPTS: PARALLEL PROCESSING AND THREADING

The idea of parallel processing is not new. It is the ancient management concept of assigning several people to work on the same task at the same time. The idea is that if one worker can dig a ditch in 4 hours, two workers can dig a ditch in about 2 hours and 4 workers can dig a ditch in about 1 hour. This is the concept behind parallel processing. If a job runs in 120 seconds on 1 CPU, it can run in 60 seconds if we can (have software and an OS that can manage the work and) run the job on two CPUs. Major sections of SAS are being re-written to split tasks and then manage tasks in parallel.

A thread can be thought of as a "worker" in the example above. It is like a small, special purpose program that does a very specific task. Such a task might be sorting part of a file, or getting part of a table from a disk drive. SAS writes these threads and hands them off to the OS to manage. A well written thread can function independently of the SAS supervisor, and other threads, and return its results to SAS for further processing. SAS creates the threads and the user gets the benefits.

However, there are limits to adding threads or workers to a task. The effect of adding threads/workers is not really linear for several reasons. The task must be capable of being divided, and once divided, the "workers/threads" must be able to function without interfering with each other. Not all tasks can be divided this way. Additionally, some time must be spent in coordinating the workers/threads.

This problems of limited returns from adding resources/threads to a task has everyday parallels. If it takes a woman 9 months to make a baby, two women can not make a baby in 4.5 months. In the ditch-digging example, if we assign so many workers to ditch digging that they start hitting each other with their shovels we have a situation where workers interfere with each other and slow the process down. Finally, the ditch digging crew will likely need a manager to iron out disputes and re-assign workers as they finish their section of the ditch. The more workers assigned to a job, the more management effort required. SAS and the O.S. take resources, and time, as they manage threads.

THE SPDE FILE STRUCTURE

The SPDE data set structure has been modified, from the base data set structure, to take advantage of multithreading. Indexed data sets have always been split into to two components - a data file (combining metadata and data) and an index file, although the user sees all as one file in SAS.

In SPDE, the data set (table) has been split into several physical files - multiple data files and metadata files, allowing threads to access the parts in parallel. The index file has also been split. The programmer must use the LIBNAME statement to locate the file parts on several different drives to reduce I/O conflicts. However, after issuing the libname statement, the programmer accesses a SPDE data set as a single entity. The operating system recognizes and maintains the different files.

The parts of a SPDE data set (table) have an file name extension of SPDS9. The parts of a SPDE data set are:

1. The metadata file: This describes the variables in the data set and all information (paths, sort information, variable attributes such as name, location, label, length, format, etc.) needed to access the various data and index parts. It must begin in the primary path (pg13). The metadata parts have a MDF as a component of the file name (as in HOPE.MDF.0.0.0.SPDS for a SAS data set named HOPE) when viewed using the operating system. Typically metadata files are small.

2. The data parts: These have a component DPF and should be spread (partitioned) over several drives and I/O channels. SAS will spawn threads (up to a number determined by the number of CPUs, the number of partitions, the value of the THREADNUM= option, the value of the SPDEMAXTHREADS= option and the operating system) to read the data in parallel. These files can be very big.

3. SPDE indexes: These are composed of two types of physical files - the global index file and the segment index file. The first, or global, part (component = HBXKEY) is a binary tree that points to the locations, in the data file, of any observations with unique values of the key/index variable(s). If the key variables(s) are not unique then it points to the second index file (e.g. customers in CA with an index on state). This second, or segment, index (component = IDXKEY) file uses bitmapping to return the locations of the desired observations. The IDX file is bitmapped and contains pointers to values that occur multiple times. This two part index has been used in some relational databases. Index files can be small to medium size.

Figure 2 contains screen prints of sample PSDE data sets Created by the code shown below.

```
Libname SPSales
          SPDE "C:\SPDE_Sales_Metadata"
          datapath=("E:\SPDE_Sales_data_E" "F:\SPDE_Sales_data_F" )
                indexpath=("G:\SPDE_sales_Indx_G")
                partsize=64;
    run;

Libname SPMktng
          SPDE "C:\Mkt_MD"
          datapath=("E:\MktDta_E"  "F:\MktDta_F")
                indexpath=("G:\MktIndx_G")
                partsize=128;
    run;

*Create a sales test file - partition size 64 meg;
data SPSales.Sales_Tests(drop=i index=(key));
    retain V2-V64 1234567. ;
    do i=1 to 1000000;
      key=i;
     output;
    end;
    run ;

*Create a marketing test file- partition size 128 meg;
data SPMktng.Marketing_Tests(drop=i index=(key));
    retain V2-V64 1234567. ;
    do i=1 to 1000000;
      key=i;
     output;
    end;
    run ;

*show ease of merging SAS data file and SPDE file;
data new;
set sashelp.class;
key = _n_;
run;

data Hope;
merge WORK.new(in=N) SPMktng.Marketing_Tests;
by key;
if N;
run;



*Converting a dataset via copying;
PROC COPY IN=WORK OUT=SPMktng;
SELECT HOPE;
RUN;
QUIT;
```

Describe a SPDE data set

Describe a SPDE data set

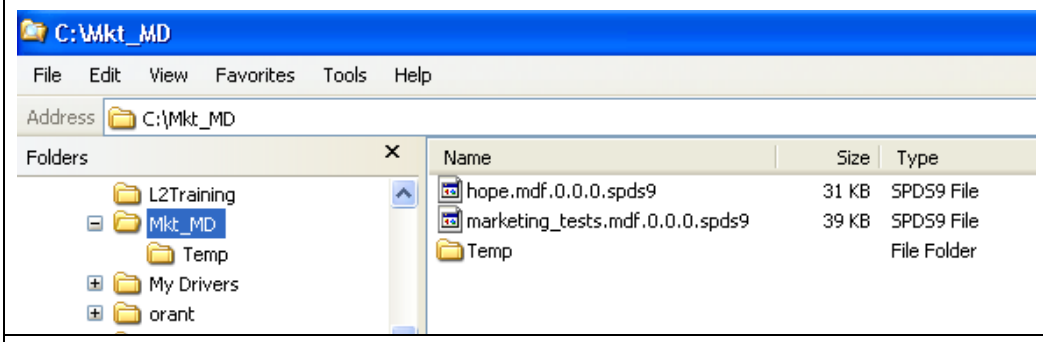
Load the data into the data set

Load the data into the data set

Merge a base SAS and a SPDE data set

Convert From base structure to SPDE structure via Proc Copy

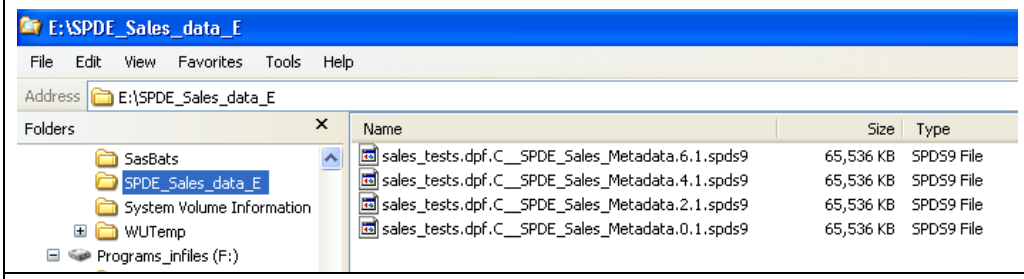
SCREEN PRINTS OF SPDE FILES FOR CODE ABOVE



METADATA

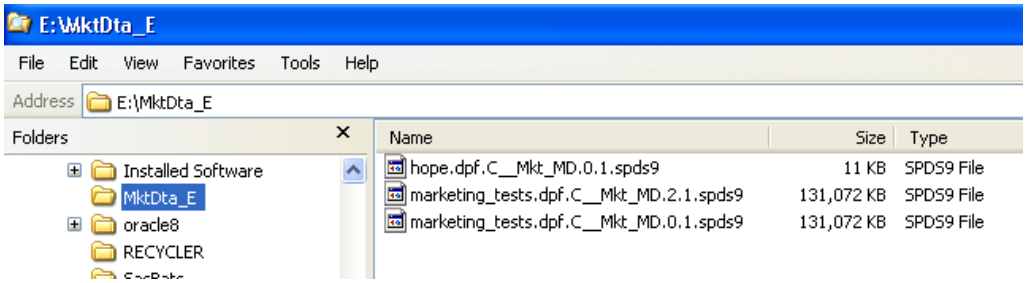
NOTE THE FILENAME COMPONENTS IN THESE SCREEN PRINTS.

MDF, DPF, IDXKEY, HBXKEY



A DATA PATH CONTAINING DATA PARTITIONS.

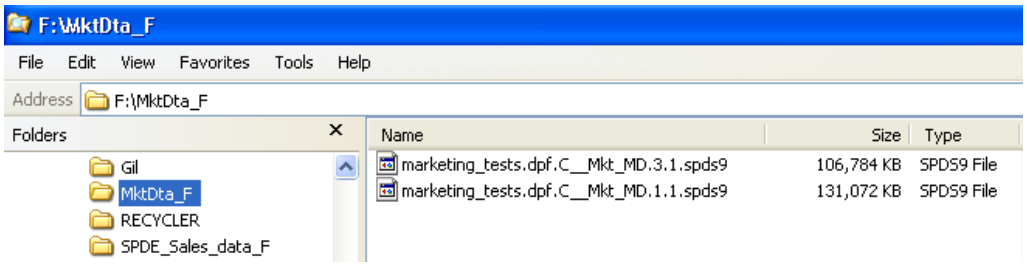
NOTE SIZE OF COMPONENTS. CHECK THE LBNAME STATEMENT FOR THIS FILE.



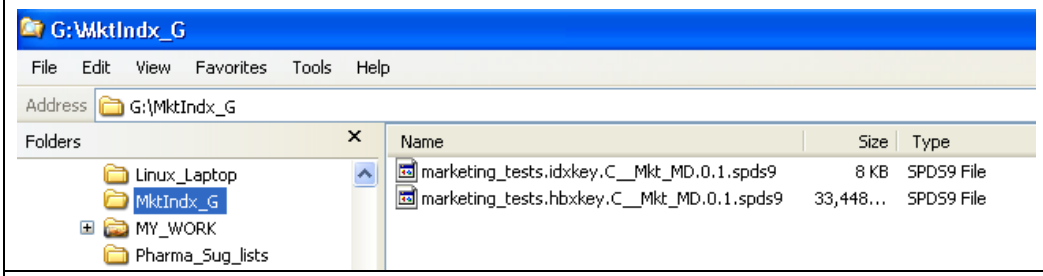
NOTE THAT THIS FOLDER CONTAINS TWO FILES.

HOPE (TOO SMALL TO BE PARTITIONED)

AND MARKETING TESTS (LARGE ENOUGH TO BE PARTITONED)



THE PARTITION SIZE OF THE FILES SHOULD BE SET SO THAT THREE OR FOUR PARTITIONS ARE IN EACH DATA (DIRECTORY). THE NUMBER OF DATA PARTITIONS PER DATA PATH SHOULD NOT EXCEED 10.



THIS SHOWS THE NEW TWO-PART INDEX FOR THE SPDE DATA SET.

NOTE

IDXKET AND HBXKEY FILES

FIGURE 2

COMPLEX SPDE SYNTAX

Using this new tool to manage data sets requires more knowledge than required for managing V8 SAS data sets. The libname statement can become quite complex. A complex, SPDE libname statement is shown below: (There is no graphic in the paper for the following libname.)

```
Libname SalesSpde  spde  'C:\metdatSales'
              /*the name of the libref and the primary path (storage area for metadata) */
       metapath='C:\OMetdatSales'
              /*overflow storage area for metadata */
       datapath=('D:\SalesDat1'   'F:\SalesDat2 ')
              /*storage area for data segments- segments are written in round robin order*/
       partsize=64
              /*the size of a data segment.in MEGS*/
       indexpath=('E:\SalesInd1' 'E:\ SalesInd 2 ')
              /*paths for the index (HDX and IDX  file types */
       bysort=yes      /*toggles on/off if SPDE sorts data for a by */
       temp=yes
              /*specifies creating of a Temp library under Primary Directory
              -This is like a SPDE work dir - allows use of 1 part filenames*/
       compress=no
              /*determines if SPDE compresses data or not – SPDE compresses by block*/
       firstobs=
              /*Tells SPDE the first obs to be processed*/
       endobs=
              /*Tells SPDE the last obs to be processed.  Good for testing*/;

   Options user=SalesSpde;
              /*This, in combo with  temp=yes , allows use of 1 part file names */
```

Many options shown above can be used as data set options or systems options. See the manual for information about how SAS handles conflicts in settings between system and data set options.

Here is an example of using a SPDE option as a Data Set Option.

```
   Proc datasets  lib=SalesSpde ;
      modify SalesQ1(ASYNCINDEX=YES);
             index create state county;  Run;Quit;
```

DATA SET OPTIONS FOR SPDE

AsyncIndex=	toggles if SPDE creates multiple indexes in parallel.
ByNoEquals=	if thisis no, observations are delivered in the same order they were in the source data set
IDXWHERE=	Specifies that indexes are to be used with where clauses
IOBLOCKSIZE=:	Specifies number of obs to be compressed at one time. This also determines the size of the I/O buffer.
SegSize=:	Specifies the number of rows represented by an index segment default is 8192
Padcompress=	Tells SPDE how many bytes to add to compression blocks when updating a data set
SsncAdd=	Yes tells SPDE to process one observation at a time- use with Uniquesave
threadnum=	Upper limit on how many threads SPDE can create when processing a SPDE data set (default = the system option SPDEMaxThreads (if you set it) or two times the number of CPUs)
uniqesave=	Tells SPDE , when updating unique indexes, to save non-unique keys to a file
Noindex=	Tells SPDE to not use any indexes when processing a where.
WhereNoIndex =	Tells SPDE to ignore a list of indexes

SPDE SYSTEM OPTIONS FOR SPDE

Segsize=	Sets the number of rows that are stored in the bitmap section of the index
MaxSegRatio=	Controls the percent of segments that are examined before the Where Planner decides on index use
MinPartSize=	Default=16 Meg. Tells SPDE the min. partition size
SPDEIndexSortSize= Default=32Meg	Sets maximum memory to be used by the sorting operation that is part of index creation.
SPDEMaxThreads=	Sets the maximum for the number of threads that SPDE can use for I/O.
SPDESortSize=	Tells SPDE how much memory it can use for sorting files. Parallel sorting takes lots of memory.
SPDEUtilLoc=	Allocates work file "utility disk space locations" (space for files created in implicit by group processing and index creation). Can be multiple locations
SPDEWhEval=	Specifies criteria that Where Planner uses for where evaluation. Default is cost.
MsgLvl=	Displays the results of the SPDE where clause planner

TUNING ISSUES AND CONCEPTS

The SPDE engine works best with a RAID (Redundant Array of Inexpensive Drives) drive system and the SPDE engine creates it's own RAID zero file system. The partitioned data set in Figure 1 is a data set that has been "striped over multiple drives". It is a "RAID zero" type of file. If the user does create/buy a more complex RAID system the metadata should be mirrored. SAS recommends against implementing RAID with software.

However the data area is setup, it should be configured so that it can deliver lots of data to the CPUs. It should deliver parallel data streams to the CPUs and the location of the data files is crucial to performance. The files should be located in such a way as to minimize I/O contention. The partition size of the files should be set so that three or four partitions are in each data path (directory). The number of data partitions per data path should not exceed 10. To calculate the partition size: Start with the estimated file size (in megs), divide by number of paths*4 to get a partition size. Round up to a power of 2. (16 Meg, 32 Meg, 64 Meg).

On a computer, with N CPUs, you should set up the data to have N independent paths to the data. It is a good idea to have one I/O controller per data path.

The Index area, SAS strongly suggests, should be striped across multiple disks. This allows parallel processing for both creating and applying the index. Note that parallel index creation/updating requires more `temporary` disk space than the same process on a single index.

SPDE uses the base SAS engine work area unless instructed not to. SPDE's work area should be configured (by using the SPDEUtilLoc= option) to send the work file to multiple drives to reduce I/O contention. SAS recommends striping, the SPDE work area.

SAS ships free (and describes on page 85 of the "Quick Guide to the SPDE Disk I/O setup) a macro that helps in tuning the SPDE engine.

CONCLUSION

SPDE can have significant effect on run times if is properly set up. Learning SPDE can allow programmers to reduce run times and make additional contributions to their company. Using SPDE can allow programmers using SAS software to successfully compete with programmers using other applications.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. You can contact the authors at:

Ian Whitlock, Independent Contractor Email: Ian.whitlock@comcast.net
Russell Lavery, Independent Contractor Email: russ.lavery@verizon.net