Paper 098-2008

Smoke and Mirrors!!! Come See How the _INFILE_ Automatic Variable and SHAREBUFFERS Infile Option Can Speed Up Your Flat File Text-Processing Throughput Speed

William E Benjamin Jr, Owl Computer Consultancy LLC, Phoenix, AZ

ABSTRACT

Processing speed for small files is never really a problem; one eye blink looks and feels much the same as two or three blinks of the eye. With today's computer speeds any machine will do a lot of work in three blinks of the eye. However, when processing hundreds of thousands, or millions of records blinks can become pauses, yawns, naps, or even an over-night sleepover. When a Base SAS® data step reads in text files and writes out nearly the same file the work proceeds as follows; (a) read input file data into input buffer, (b) move data values to data step variables (c) process data variables, (d) move data to output buffer, (e) write output file from output buffer. SAS reads your source code every time you run the code and builds the "current" processing steps (unless the code is compiled separately first) then executes them. By packing more work into each instruction less time is wasted between instructions, this means more work in less real time. The SHAREBUFFERS option eliminates moving data from the input to the output buffer (a BIG time saver). And the _infile_ automatic variable can be used in function calls on either side of the "=" sign. This eliminates some or all of the data step variable processing like this; (a) read input file data into a common buffer, (b) process data values, (c) write data output text file from common buffer. The resulting time saved can result in faster execution of the program.

INTRODUCTION

This paper will examine one task done three ways with and without using the SHAREBUFFERS option. The code will be explained and the results shown. Since, one task performed on a computer is not always going to run in the same amount of time, job averages for 10 executions of the same task will be displayed. This is because the background tasks that are executing are not easily controlled. Additionally, the SAS macros that were used to execute the code seemed to have some extra time used on the first run. So, that run was excluded and 10 other passes were made to be included in the averages. The SHAREBUFFERS option speed needs one I/O file.

SPEED, SPEED, and More SPEED …

The bane of all programmers is speed, how fast can you write the program, how fast will it run, how soon can I have my report, how fast can … . Finding the answers to those questions and implementing solutions turns a programmer into a good programmer, a good programmer into a great programmer, and a great programmer into a consultant. One of the tricks of the trade to do more, faster is by doing less with each record. Often in the data processing world a company will use more than one system to hold and update data for individual systems. Payroll will be processed by COBOL, retirement on SAS, survey questions in Access, sales reports in Excel and on and on. The link between these programs is often a data file passed between them.

The Data - Random Input Data File Generation

In simple terms, a test file was created with 500,000 records each having 510 bytes (250MB total file size). All of the data was randomly created one byte at a time. The ASCII character set was used to create the data file, and only printable characters were allowed into the file. The random generation of the data produced a data file with only a few records containing a "N" in byte 23. In this case there were 12658 records with "N" in column 23, this was 2.53% of the data. The input file with the carriage return and line feed (cr/lf) added by the PUT command is an exact multiple of the Operating System disk block size. This reduces the actual system overhead used in the I/O processing. This was done in an effort to isolate system effort from the SAS run-time work. The time used was 3:14:16 real time and 3:03:00 User CPU seconds. The author did not determine the extent of the impact potential of using other data block sizes.

The following code generated the input text file.

```
data _Null_;                                          *No output SAS file;
file in_text notitles noprint linesize=510;           *Define the output disk file;
b = ' ';                                              *temp variable;
attrib text length = $ 510;                           *Define the output record;
    do i = 1 to 500000;                               *Define the number of output records;
        k = 0;                                        *Index points to next output character;
        do j = 1 to 510;                              *Write 510 characters per line (plus cr/lf);
            k+1;                                      *point to the next output character;
            do until(((a>47) and (a<58)) or ((a>64) and (a<91)) or ((a>96) and (a<123)));
                a = int(ranuni(01261950)*100);        *get a new random character;
                if (((a>47) and (a<58)) or ((a>64) and (a<91)) or ((a>96) and (a<123))) then  do;
                    b = put(byte(a),$char1.);         *convert random number to a character;
                    substr(text,k,1) =  b;            *save a valid character;
                end;                                  *end of loop that saves a valid character;
            end;                                      *end of search for one valid character;
        end;                                          *one full line complete;
        put text;                                     *output one line (the put adds a cr/lf);
    end;                                              *end of loop to write 500,000 records;
run;
```

The Warning – Some of the Following Methods Modify the Source file !!!

Yes, just like TV commercials which warn of all sorts of bad side effects, some of the following WILL modify the INPUT file.

The Task

Network interfaces between systems are becoming more standardized and therefore easier for a programmer to tap into when the need arises. Complex system integration is usually not attempted for the sake of answering the "How Fast" questions. If one system can write a "Flat File" or a "Text File" that the next system can read, then the integration problem is "SOLVED" and the team moves on. Often programmers are asked to "ADD" some data to a flat file and send it somewhere else for the final or next processing step. Say, that byte 24 and byte 25 have to be swapped, only if byte 23 was an "N". The examples in this paper do just that, read in the variables with an input statement (so no data is lost), make the variable value changes, and write out the data with a "PUT" statement. The task is simple but the methods vary. Three different methods appear in this paper, each with three input/output file structures. Furthermore, to eliminate manual intervention (and processing differences) the jobs are encased in macros. Every effort will be made to highlight the code, and hide the macros. Data _NULL_ steps were used to remove from the testing any effort by the data step to attempt to write an output SAS file, thereby saving execution time in a effort to measure only the test items.

Method 1- Full variable definition of input/output files

The first method is to fully define each variable in the input and output files. This is done with full definitions in the "Input" and "Put" commands. The variable MY_N is tested for a value of "N", and when found swap_1 is moved into a temporary variable, swap_2 is moved to swap_1, and then the temporary variable value is returned to swap_2. The temporary variable was used to in the swap routine to hold intermediate data. See the SAS Macros RUN_IT1, RUN_IT4, and RUN_IT7 below. A limited number of variables were defined to isolate the intended work to be measured and fully define the input and output files with large text variables.

```
 input
@0001  var1   $char22.     @0023  my_N   $char1.     @0024  swap_1 $char1.     @0025  swap_2 $char1.
@0026  text1  $char175.    @0201  text2   $char150.  @0351  text3    $char160.;

if my_n = "N" then do;
   temp = swap_1;
   swap_1 = swap_2;
   swap_2 = temp;
end;

 put
@0001  var1   $char22.     @0023  my_N   $char1.     @0024  swap_1 $char1.     @0025  swap_2 $char1.
@0026  text1  $char175.    @0201  text2  $char150.   @0351  text3  $char160.;
```

Method 2- No variable definition of input/output files

The second method is to not define any variable in the input and output files. This is done with by referencing substrings of the SAS Automatic variable "_INFILE_". Byte 23 was tested for an "N", and when found, byte 24 was saved in a SAS variable called TEMP. Next byte 25 of the _INFILE_ variable was moved directly to byte 24 and byte 24 was copied from TEMP back into _INFILE_ byte 25. The temporary variable was used to in the swap routine to hold intermediate data. See the SAS Macros RUN_IT2, RUN_IT5, and RUN_IT8 below. One variable was defined, and the rest of the work was done with function calls.

```
Input;
    if (substr(_infile_,23,1)) = "N" then do;
       temp = substr(_infile_,24,1);
       substr(_infile_,24,1) = substr(_infile_,25,1);
       substr(_infile_,25,1) = temp;
    end;
put _infile_;
```

Method 3- Hybrid Solution, Define only input variables

The third method is to define only variables in the input file and write the output file directly from the "_INFILE_" automatic variable. This is done by referencing output substrings of the SAS Automatic variable "_INFILE_". The variable MY_N was tested for an "N", and when found, swap_2 was copied directly to byte 24 of the SAS Automatic variable "_INFILE_". Then swap_1 was saved in byte 25 of the SAS Automatic variable "_INFILE_". No temporary variable was used to hold intermediate data. See the SAS Macros RUN_IT3, RUN_IT6, and RUN_IT9 below. Three variables were defined, and the rest of the work was done with function calls.

```
input
   @0023  my_N $char1.      @0024  swap_1 $char1.     @0025  swap_2 $char1.;

   if my_n = "N" then do;
      substr(_infile_,24,1) = swap_2;
      substr(_infile_,25,1) = swap_1;
   end;
put _infile_;
```

Input / Output File Structures

The options and file name definitions are defined as close to the top directory level as possible to reduce O/S overhead. The disk was NTFS and compressed, but most systems compress their disk farms to increase storage. There is of course a processing hit for the trade-off of compression vs. speed, but this author is willing to accept that as a necessary space requirement. One input data file of random characters was created and used for all of the testing. Method one wrote all new files without changing the original input file, for the other tests the files were created and then the disk was defragmented so that all of the files would be working from files that were nearly exactly the same in both content and the number of fragments in the initial files. These tests required 11 executions of the same code, and the testing was stopped between each test set to defragment the disk so each test could have similar starting conditions. The Log files were written to a separate file on a different disk for each test.

```
options fullstimer mprint mlogic symbolgen;
%let rec_size = 510;
filename in_text    'f:\old_random_characters.txt';
filename out_text  'f:\new_random_characters.txt';

filename samefila 'f:\tst_random_characters_a.txt';
filename samefilb 'f:\tst_random_characters_b.txt';
filename samefilc 'f:\tst_random_characters_c.txt';

filename shr_txta  'f:\tst_random_characters_a_Sharebuffer.txt';
filename shr_txtb  'f:\tst_random_characters_b_Sharebuffer.txt';
filename shr_txtc  'f:\tst_random_characters_c_Sharebuffer.txt';
```

The first file structure uses one input file that all three of the Processing methods use. See macros Run_it1, Run_it4, and Run_it7. These macros read a source file and each task writes out a new output file with a new name. All output files have the same name but new disk space is used for each output file. And the old space is recovered by the Operating System at the end of the SAS Data step. The output file characteristics do not change for this processing as it does for the other tests and therefore no additional filenames are required.

```
infile in_text linesize=&rec_size truncover;
file  out_text linesize=&rec_size;
```

The second file structure used by macros Run_it2, Run_it5, and Run_it8 requires only one file name. Therefore this file must be refreshed between each execution before the start of each Data step; a fresh copy of the Object (input and output) data file is created. Since the Data step only uses one file name for both input and output, each execution of the Data step must have a new copy of the data file. Note – the x in the file name stands for an a, b, or c. This was used to ensure that no overlap of defined file characteristics occurred in the processing.

```
infile samefilx linesize=&rec_size truncover ;
file    samefilx linesize=&rec_size;
```

The third file structure used by macros Run_it3, Run_it6, and Run_it9 requires only one file name. Therefore this file must be refreshed between each execution before the start of each Data step; a fresh copy of the Object (input and output) data file is created. Since the Data step only uses one file name for both input and output, each execution of the Data step must have a new copy of the data file. The difference between structure two and three is that the data step itself uses shared internal buffers. Note – the x in the file name stands for an a, b, or c. This was used to ensure that no overlap of defined file characteristics occurred in the processing.

```
infile shr_txtx linesize=&rec_size truncover sharebuffers;
file    shr_txtx linesize=&rec_size;
```

The Computer System

First a little about the machine, the code was executed on a X86 Family 15 model 12 Stepping 0 AuthenticAMD ~2009 MHZ CPU running Microsoft® Windows/XP Professional using SAS V9.1 (TS1M3). In other words, the standard run-of-the-mill four year old home computer. The computer was equipped with 1GB of real memory and a 2GB virtual memory page file. Over ½ of the real memory was available at run-time. The data was stored a logical drive on an extended partition of 20GB NTFS compressed local disk drive, and the full drive size was 28GB. The logical drive and the system drive were on the same physical disk drive. A 512 byte block-size was used. Three computers were networked together, but no data transferring was occurring during the testing. These tests were run after the disk had been defragmented so that the available disk space would be in the largest possible chunks (a technical term?) The target drive had 70% of the disk available as free space. The SAS autosave option was turned off during testing.

It should be noted that not all computer systems use the same disk formats. Most PC's use the format described here, however individual machines may have different disk block sizes for the disk formats. Most Mainframes (using EBCDIC data encoding) will have larger disk block sizes with some over 40K bytes per disk block. The mainframes also have several methods on creating data blocks. These include fixed length data blocks with no cr/lf, and variable length data records that include four bytes to describe the actual size of the records/data blocks. Most Unix and Linux systems may use 512 as a disk block size, but contact your system administrator to get the exact size.

The Testing

For the testing the Base SAS runtime options FULLSTIMER, MPRINT, MLOGIC, and SYMBOLGEN were used to record the Wall (execution) time, the User CPU time, and the System CPU time used by the processing, and also to display the processing as it progressed throughout the tests. All of the tests were run in a clean environment, the disk had just been defragmented, The SAS session was just started, and the SAS autosave option was turned off. There were no other foreground tasks running and there was no network traffic looking for data on the computer running the tests. In short, the test bed was as clean as it could be made for a home computer. And all tests faced the same environment. Tests that appeared to be impacted by unknown system actions were re-executed in their entirety. Any given unit of 10 executions of the same test on the same quiet system should be expected to give approximately the same results for the same effort, therefore, any test unit with a result that differed by more than 30% in any given instance of the test trial was rejected and the test unit was re-run.

The Results – Graphically

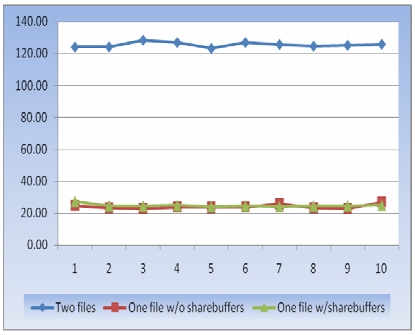
The following graphs show in pictures the results the test suite. Three methods and three file structures showing three test measures. Method 1 tests all fully define the input and output records and the graphs show the results for testing with (1) two files, (2) One file without a reference to the SAS SHAREBUFFERS infile option, but using the same file name for input and output, and (3) One file with a reference to the SAS SHAREBUFFERS infile option, using the same file name for input and output. Other input/output configurations were not tested because SAS documentation describes the SHAREBUFFERS option as only being effective when the same file is used for both input and output.

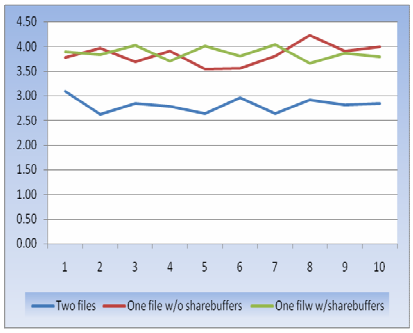
Method 2 tests are structured to only use the SAS automatic variable _INFILE_ as the working storage of the program, choosing to perform all of the work using function calls, rather than defining data variables. However the Method 3 tests are a hybrid of the first two steps. The input variables are defined in the input statement and the _INFILE_ automatic variable is used for the output. Notice that this works with all three input / output file structures. Additionally the Method 3 process does not need a temporary variable to perform the swap of the data bytes 24 and 25.

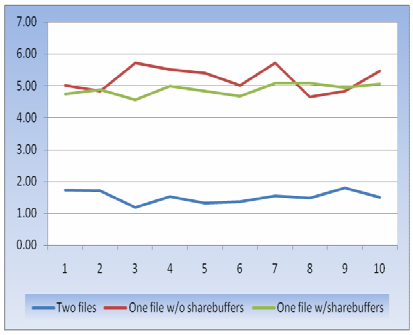
The graphs clearly point out that the User CPU and System CPU time increases with Methods 2 and 3, but the time that your wait time for the tasks to finish decreases with these methods. All times listed below are in seconds.

Method 1 – Test results

Wall Time	User CPU Time	System CPU Time



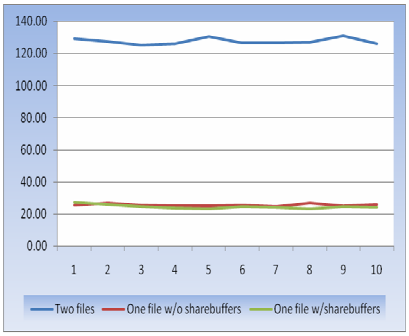


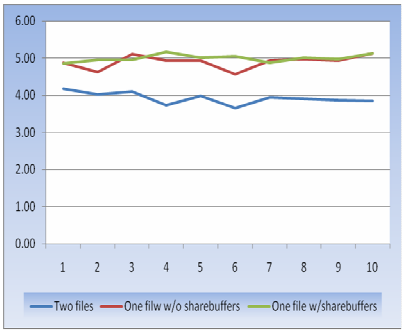


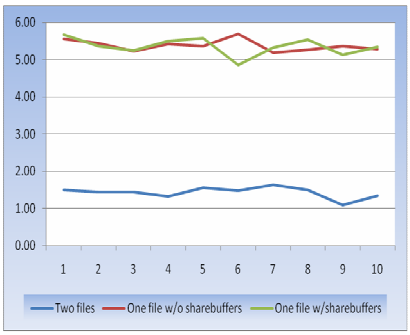
File structures 2 and 3 decrease wall time when the file is fully defined, and increase User and System CPU times at the same time.

Method 2 – Test results

Wall Time	User CPU Time	System CPU Time



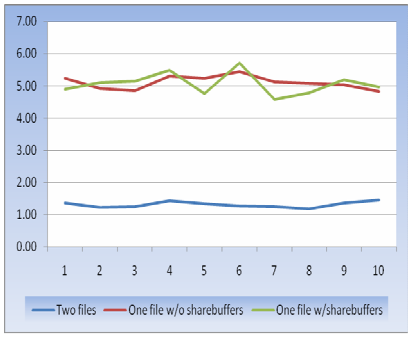




File structures 2 and 3 decrease wall time when the file is not fully defined and only funftions are used to update the output data, and increase User and System CPU times at the same time.

Method 3 – Test results

Wall Time	User CPU Time	System CPU Time



File structures 2 and 3 decrease wall time when the hibrid Method 3 is used to update the output data, and increase User and System CPU times at the same time.

The Code

The following SAS code processed the data to produce SAS logs that were scanned for the data graphed above. Your results will vary from the results shown above because your system may not be configured exactly the same was as the test platform. However, this author does not expect your results to be dramatically different that what is submitted here. The code shown here is in two columns on several pages.

```
 options fullstimer mprint mlogic symbolgen;
 %let rec_size = 510;

 filename in_text   'f:\old_random_characters.txt';
 filename out_text  'f:\new_random_characters.txt';

 filename samefila  'f:\tst_random_characters_a.txt';
 filename samefilb  'f:\tst_random_characters_b.txt';
 filename samefilc  'f:\tst_random_characters_c.txt';

 filename shr_txta
'f:\tst_random_characters_a_Sharebuffer.txt';
 filename shr_txtb
'f:\tst_random_characters_b_Sharebuffer.txt';
 filename shr_txtc
'f:\tst_random_characters_c_Sharebuffer.txt';

 filename my_log1   'c:\coders_corner_test_1_log_file.txt';
 filename my_log2   'c:\coders_corner_test_2_log_file.txt';
 filename my_log3   'c:\coders_corner_test_3_log_file.txt';

 filename my_log4   'c:\coders_corner_test_4_log_file.txt';
 filename my_log5   'c:\coders_corner_test_5_log_file.txt';
 filename my_log6   'c:\coders_corner_test_6_log_file.txt';

 filename my_log7   'c:\coders_corner_test_7_log_file.txt';
 filename my_log8   'c:\coders_corner_test_8_log_file.txt';
 filename my_log9   'c:\coders_corner_test_9_log_file.txt';
```

```
/*
* this code is run once to create a sample text file ;
data _Null_;
file in_text notitles noprint linesize=510;
b = ' ';
attrib text length = $ 510;
   do i = 1 to 500000;
      k = 0;
      do j = 1 to 510;
         k+1;
         do until(((a>47) and (a<58)) or ((a>64) and (a<91)) or
((a>96) and (a<123)));
            a = int(ranuni(01261950)*100);
            if (((a>47) and (a<58)) or ((a>64) and (a<91)) or
((a>96) and (a<123))) then  do;
               b = put(byte(a),$char1.);
               substr(text,k,1) =  b;
            end;
         end;
      end;
      put text;
   end;
run;

  * job to find out how many "N"s in Column 23;
 data _null_;

infile in_text linesize=&rec_size truncover sharebuffers;
  file shr_text linesize=&rec_size;

input
   @0023   my_N $char1.
   @0024   swap_1 $char1.
   @0025   swap_2 $char1.  ;

   if my_n = "N" then do;
      substr(_infile_,24,1) = swap_2;
      substr(_infile_,25,1) = swap_1;
      put _infile_;
   end;
run;

*/
%macro copy_it(out_file);
data _null_;
   infile in_text   linesize=&rec_size truncover sharebuffers;
     file &out_file linesize=&rec_size;
   input;
   put _infile_;
run;
%mend copy_it;
%macro decrement;
data _null_;
   loops = &loops;
   loops = loops - 1;
   call symput('LOOPS',loops);
run;
%mend decrement;
************************************************;
%macro run_it1;
  proc printto log=my_log1 new;
  data _null_;

infile in_text linesize=&rec_size truncover;
  file out_text linesize=&rec_size;

input
   @0001   var1   $char22.
   @0023   my_N   $char1.
   @0024   swap_1 $char1.
   @0025   swap_2 $char1.
   @0026   text1  $char175.
   @0201   text2  $char150.
   @0351   text3  $char160.     ;

   if my_n = "N" then do;
      temp = swap_1;
      swap_1 = swap_2;
      swap_2 = temp;
   end;

   put
   @0001   var1   $char22.
   @0023   my_N   $char1.
   @0024   swap_1 $char1.
   @0025   swap_2 $char1.
   @0026   text1  $char175.
   @0201   text2  $char150.
   @0351   text3  $char160.     ;

run;

%let loops = 10;
%do %until(&loops=0);
data _null_;

infile in_text linesize=&rec_size truncover;
  file out_text linesize=&rec_size;

input
   @0001   var1   $char22.
   @0023   my_N   $char1.
   @0024   swap_1 $char1.
   @0025   swap_2 $char1.
   @0026   text1  $char175.
   @0201   text2  $char150.
   @0351   text3  $char160.;

   if my_n = "N" then do;
      temp = swap_1;
      swap_1 = swap_2;
      swap_2 = temp;
   end;

   put
   @0001   var1   $char22.
   @0023   my_N   $char1.
   @0024   swap_1 $char1.
   @0025   swap_2 $char1.
   @0026   text1  $char175.
   @0201   text2  $char150.
   @0351   text3  $char160.;

run;
%decrement;
%end;
%mend run_it1;

************************************************;
%macro run_it2;
  proc printto log=my_log2 new;
data _null_;
   infile samefila linesize=&rec_size truncover ;
     file samefila linesize=&rec_size;

input
```

```
    @0001   var1   $char22.
    @0023   my_N   $char1.
    @0024   swap_1 $char1.
    @0025   swap_2 $char1.
    @0026   text1  $char175.
    @0201   text2  $char150.
    @0351   text3  $char160.;

    if my_n = "N" then do;
       temp = swap_1;
       swap_1 = swap_2;
       swap_2 = temp;
    end;

    put
    @0001   var1   $char22.
    @0023   my_N   $char1.
    @0024   swap_1 $char1.
    @0025   swap_2 $char1.
    @0026   text1  $char175.
    @0201   text2  $char150.
    @0351   text3  $char160.;

run;
%let loops = 10;
%do %until(&loops=0);

%copy_it(samefila);

data _null_;
    infile samefila linesize=&rec_size truncover ;
      file samefila linesize=&rec_size;

input
    @0001   var1   $char22.
    @0023   my_N   $char1.
    @0024   swap_1 $char1.
    @0025   swap_2 $char1.
    @0026   text1  $char175.
    @0201   text2  $char150.
    @0351   text3  $char160.;

    if my_n = "N" then do;
       temp = swap_1;
       swap_1 = swap_2;
       swap_2 = temp;
    end;

    put
    @0001   var1   $char22.
    @0023   my_N   $char1.
    @0024   swap_1 $char1.
    @0025   swap_2 $char1.
    @0026   text1  $char175.
    @0201   text2  $char150.
    @0351   text3  $char160. ;

run;

%decrement;
%end;
%mend run_it2;

************************************************;
%macro run_it3;
  proc printto log=my_log3 new;

data _null_;
    infile shr_txta linesize=&rec_size truncover sharebuffers;
      file shr_txta linesize=&rec_size;

input
    @0001   var1   $char22.
    @0023   my_N   $char1.
    @0024   swap_1 $char1.
    @0025   swap_2 $char1.
    @0026   text1  $char175.
    @0201   text2  $char150.
    @0351   text3  $char160.;

    if my_n = "N" then do;
       temp = swap_1;
       swap_1 = swap_2;
       swap_2 = temp;
    end;

    put
    @0001   var1   $char22.
    @0023   my_N   $char1.
    @0024   swap_1 $char1.
    @0025   swap_2 $char1.
    @0026   text1  $char175.
    @0201   text2  $char150.
    @0351   text3  $char160. ;

run;
%let loops = 10;
%do %until(&loops=0);

%copy_it(shr_txta);

data _null_;
    infile shr_txta linesize=&rec_size truncover sharebuffers;
      file shr_txta linesize=&rec_size;

input
    @0001   var1   $char22.
    @0023   my_N   $char1.
    @0024   swap_1 $char1.
    @0025   swap_2 $char1.
    @0026   text1  $char175.
    @0201   text2  $char150.
    @0351   text3  $char160. ;

    if my_n = "N" then do;
       temp = swap_1;
       swap_1 = swap_2;
       swap_2 = temp;
    end;

    put
    @0001   var1   $char22.
    @0023   my_N   $char1.
    @0024   swap_1 $char1.
    @0025   swap_2 $char1.
    @0026   text1  $char175.
    @0201   text2  $char150.
    @0351   text3  $char160. ;

run;

%decrement;
%end;
%mend run_it3;

************************************************;
```

```
%macro run_it4;
  proc printto log=my_log4 new;
data _null_;

infile in_text linesize=&rec_size truncover sharebuffers;
  file shr_text linesize=&rec_size;

input ;

    if (substr(_infile_,23,1)) = "N" then do;
        temp = substr(_infile_,24,1);
        substr(_infile_,24,1) = substr(_infile_,25,1);
        substr(_infile_,25,1) = temp;
    end;

    put _infile_;

run;
%let loops = 10;
%do %until(&loops=0);
data _null_;

infile in_text linesize=&rec_size truncover sharebuffers;
  file shr_text linesize=&rec_size;
input;
    if (substr(_infile_,23,1)) = "N" then do;
        temp = substr(_infile_,24,1);
        substr(_infile_,24,1) = substr(_infile_,25,1);
        substr(_infile_,25,1) = temp;
    end;

    put _infile_;
run;
%decrement;
%end;
%mend run_it4;

************************************************;
%macro run_it5;
  proc printto log=my_log5 new;
  data _null_;
    infile samefilb linesize=&rec_size truncover ;
      file samefilb linesize=&rec_size;
input;
    if (substr(_infile_,23,1)) = "N" then do;
        temp = substr(_infile_,24,1);
        substr(_infile_,24,1) = substr(_infile_,25,1);
        substr(_infile_,25,1) = temp;
    end;

    put _infile_;

run;
%let loops = 10;
%do %until(&loops=0);

%copy_it(samefilb);

data _null_;
    infile samefilb linesize=&rec_size truncover ;
      file samefilb linesize=&rec_size;
input;
    if (substr(_infile_,23,1)) = "N" then do;
        temp = substr(_infile_,24,1);
        substr(_infile_,24,1) = substr(_infile_,25,1);
        substr(_infile_,25,1) = temp;
    end;

    put _infile_;

run;

%decrement;
%end;
%mend run_it5;

************************************************;
%macro run_it6;
   proc printto log=my_log6 new;
  data _null_;
    infile shr_txtb linesize=&rec_size truncover sharebuffers;
      file shr_txtb linesize=&rec_size;
input;
    if (substr(_infile_,23,1)) = "N" then do;
        temp = substr(_infile_,24,1);
        substr(_infile_,24,1) = substr(_infile_,25,1);
        substr(_infile_,25,1) = temp;
    end;

    put _infile_;

run;
%let loops = 10;
%do %until(&loops=0);

%copy_it(shr_txtb);

data _null_;
    infile shr_txtb linesize=&rec_size truncover sharebuffers;
      file shr_txtb linesize=&rec_size;
input;
    if (substr(_infile_,23,1)) = "N" then do;
        temp = substr(_infile_,24,1);
        substr(_infile_,24,1) = substr(_infile_,25,1);
        substr(_infile_,25,1) = temp;
    end;

    put _infile_;

run;

%decrement;
%end;
%mend run_it6;

************************************************;
%macro run_it7;
  proc printto log=my_log7 new;
data _null_;

infile in_text linesize=&rec_size truncover;
  file out_text linesize=&rec_size;

input
    @0023   my_N $char1.
    @0024   swap_1 $char1.
    @0025   swap_2 $char1.  ;
    if my_n = "N" then do;
        substr(_infile_,24,1) = swap_2;
        substr(_infile_,25,1) = swap_1;
    end;

    put _infile_;
run;

%let loops = 10;
```

```
 %do %until(&loops=0);

 data _null_;

 infile in_text linesize=&rec_size truncover;
  file out_text linesize=&rec_size;

 input
    @0023   my_N $char1.
    @0024   swap_1 $char1.
    @0025   swap_2 $char1.  ;

    if my_n = "N" then do;
       substr(_infile_,24,1) = swap_2;
       substr(_infile_,25,1) = swap_1;
    end;

    put _infile_;
 run;
 %decrement;
 %end;
 %mend run_it7;
 ************************************************;
 %macro run_it8;
  proc printto log=my_log8 new;
data _null_;
    infile samefilc linesize=&rec_size truncover ;
      file samefilc linesize=&rec_size;
 input
    @0023   my_N $char1.
    @0024   swap_1 $char1.
    @0025   swap_2 $char1.  ;

    if my_n = "N" then do;
       substr(_infile_,24,1) = swap_2;
       substr(_infile_,25,1) = swap_1;
    end;

    put _infile_;
run;

 %let loops = 10;
 %do %until(&loops=0);

 %copy_it(samefilc);

 data _null_;
    infile samefilc linesize=&rec_size truncover ;
      file samefilc linesize=&rec_size;
 input
    @0023   my_N $char1.
    @0024   swap_1 $char1.
    @0025   swap_2 $char1.  ;
    if my_n = "N" then do;
       substr(_infile_,24,1) = swap_2;
       substr(_infile_,25,1) = swap_1;
    end;

    put _infile_;

 %decrement;
 %end;
 %mend run_it8;
 ************************************************;
 %macro run_it9;
  proc printto log=my_log9 new;

data _null_;
    infile shr_txtc linesize=&rec_size truncover sharebuffers;
      file shr_txtc linesize=&rec_size;
 input
    @0023   my_N $char1.
    @0024   swap_1 $char1.
    @0025   swap_2 $char1.  ;

    if my_n = "N" then do;
       substr(_infile_,24,1) = swap_2;
       substr(_infile_,25,1) = swap_1;
    end;

    put _infile_;
run;

 %let loops = 10;
 %do %until(&loops=0);

 %copy_it(shr_txtc);

 data _null_;
    infile shr_txtc linesize=&rec_size truncover sharebuffers;
      file shr_txtc linesize=&rec_size;
 input
    @0023   my_N $char1.
    @0024   swap_1 $char1.
    @0025   swap_2 $char1.  ;

    if my_n = "N" then do;
       substr(_infile_,24,1) = swap_2;
       substr(_infile_,25,1) = swap_1;
    end;

    put _infile_;

 %decrement;
 %end;
 %mend run_it9;

*************************************************************************;
 %run_it1;
 proc printto ;
run;
 %run_it2;
 proc printto ;
 run;
 %run_it3;
 proc printto ;
 run;
 %run_it4;
 proc printto ;
 run;
 %run_it5;
 proc printto ;
 run;
 %run_it6;
 proc printto ;
 run;
 %run_it7;
 proc printto ;
 run;
 %run_it8;
 proc printto ;
 run;
 %run_it9;
 proc printto ;
 run;
```

CONCLUSION

There are dramatic differences in wait times that can be achieved by using both the SAS SHAREBUFFERS and the _INFILE_ automatic variable in your SAS code. However, this author does not recommend using it on your only or last clean copy of a dataset. The potential for error always exists and should not be taken lightly. With that said, this simple example that changed 25,316 bytes of a 250MB file has demonstrated significant differences in both speed to accomplish the job and amount and type of resources used. When you use Methods 2 or 3 this paper demonstrates that you get your job completed 5.2 times sooner, but your computer uses 1.3 times as much User CPU time and 3.5 times more System CPU resources in less time than Method 1.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author:

Name	William E Benjamin Jr
Enterprise	Owl Computer Consultancy, LLC
Address	P.O. Box 42434
City, State, ZIP	Phoenix, AZ, 85080
Work Phone:	602-942-0370
Fax:	602-942-3204
E-mail:	wmebenjaminjr3@juno.com
Web:	www.OwlFunding.com