Paper 102-2022

Simplified Linux SAS ® Log Comparison: Filtering Differences Into "Useful" and "Useless" Files

Bruce Gilsen, Federal Reserve Board, Washington, DC

ABSTRACT

SAS ® log files from two executions of the same program are frequently compared. For example, we might compare today's results to a baseline run we know is correct to test if a change to the SAS release, operating system, external data, our code, or anything else has altered the results.

Log file comparisons are often done programmatically, but it's also common to generate a line-by-line comparison with the Linux diff command and eyeball the results. This "eyeball analysis" can be difficult when programs have many steps because the diff output is often cluttered with many records that are useless to most people such as step summaries with real and/or CPU time differences and page headings with date/time differences.

The DIFFSPLIT macro runs diff for two SAS log files, parses the output with Perl Regular Expressions in a DATA step, and splits the diff output into two files: one with useful file differences, and the other with useless differences, allowing us to focus on just the differences of interest.

The macro in this paper compares files in Linux and treats step summaries with real and/or CPU time differences and page headings with date/time differences as useless. As shown in the paper, it can easily be extended to run on other platforms such as Windows or split the text using different criteria.

INTRODUCTION

SAS log files from two executions of the same program are frequently compared. For example, we might compare today's results to a baseline run we know is correct to test if a change to the SAS release, operating system, external data, our code, or anything else has altered the results.

Log file comparisons are often done programmatically, but it's also common to generate a line-by-line comparison with the Linux diff command and eyeball the results. This "eyeball analysis" can be difficult when programs have many steps because the diff output is often cluttered with many records that are useless to most people such as step summaries with real and/or CPU time differences and page headings with date/time differences.

In my case, I run a SAS test job with over 100 steps to test operating system or SAS software upgrades. When I compare SAS logs from the production and test environments, the diff command generates hundreds of records, most of which are blocks of records like the following that aren't useful to me.

Step summaries with real and/or CPU time differences.

```
394c394
<       real time           0.01 seconds
---
>       real time           0.06 seconds

414,415c414,415
<       real time           0.02 seconds
<       cpu time            0.01 seconds
---
>       real time           0.14 seconds
>       cpu time            0.02 seconds
```

```
456c456
<       cpu time            0.01 seconds
---
>       cpu time            0.00 seconds
```

Page headings with date/time differences.

```
< 7                              The SAS System 10:40 Saturday, November 7, 2020
---
> 7                              The SAS System 15:31 Saturday, November 7, 2020
```

WORK library names.

```
2051c2051
<  WORK=/data/lscratch/sas_m1xxx00/SAS_work6A980009D952_sas001.abcd.gov
---
>  WORK=/data/lscratch/sas_m1xxx00/SAS_work57F500004BB4_sas002.abcd.gov
```

Temporary file names.

```
4680c4680
<       Filename=/tmp/getf.645458.out,
---
>       Filename=/tmp/getf.19380.out,
```

Date/time stamps for files written to in DATA steps.

```
4953c4953
<       Last Modified=23Sep2021:18:06:14
---
>       Last Modified=24Sep2021:18:06:29
```

To focus on just differences of interest, I developed the DIFFSPLIT macro, which has the following steps.

- Run diff for two SAS log files.
- Parse the diff output with Perl Regular Expressions in a DATA Step.
- Split the diff output into two files: one with useful file differences, and the other with useless differences as described above.

To simplify the macro in this paper and focus on a few potentially common cases, it only runs in Linux and treats step summaries with real and/or CPU time differences and page headings with date/time differences as useless. The macro can easily be extended to run on other platforms or split the text using different criteria, as shown at the end of the paper.

In this paper, the term "compare block" refers to a group of records consisting of line number(s) where differences occur (e.g., 4680c4680 or 414,415c414,415) and the records that differ.

A SMALL EXAMPLE TO ILLUSTRATE THE DIFF COMMAND

Permanent data sets XXX.ONE and XXX.TWO normally have the following values.

```
  XXX.ONE             XXX.TWO
```

```
var1 var11         var2 var22
  1    2             11   22
  3    4             33   44
  5    6             55   66
```

The following simple program runs nightly, and the log is compared to the prior night's log with the diff command to check for problems.

```
libname xxx 'my/directory/path';
data one;
  set xxx.one;
  var111=var1+var11;
run;
proc print data=one;
run;
data two;
  set xxx.two;
  var222=var2+var22;
run;
proc print data=two;
run;
```

One night, data set TWO has a fourth observation. This kind of change - an external change to a data set - can realistically happen in a real application.

A comparison of the current and prior log with results written to the file logcompare.text by the Linux diff command is done as follows.

```
diff myprog1.log myprog1.prior.log > logcompare.txt
```

The file logcompare.txt is listed below. Please note the following.

- DATA step and PROC PRINT information in the SAS log reflecting the data changes are likely to be of interest.
- Two instances of page headers with date/time differences and multiple instances of real and/or CPU time results with small time differences are not likely to be of interest.

```
2c2
<                                                14:38 Thursday, September 23, 2021
---
>                                                14:47 Thursday, September 23, 2021
61c61
<                                                14:38 Thursday, September 23, 2021
---
>                                                14:47 Thursday, September 23, 2021
72,73c72,73
<       real time           0.01 seconds
<       cpu time            0.01 seconds
---
>       real time           0.00 seconds
>       cpu time            0.00 seconds
82c82
<       real time           0.03 seconds
---
>       real time           0.02 seconds
91,92c91,92
< NOTE: There were 3 observations read from the data set XXX.TWO.
< NOTE: The data set WORK.TWO has 3 observations and 3 variables.
---
> NOTE: There were 4 observations read from the data set XXX.TWO.
> NOTE: The data set WORK.TWO has 4 observations and 3 variables.
```

```
95c95
<       cpu time            0.00 seconds
---
>       cpu time            0.01 seconds
101c101
< NOTE: There were 3 observations read from the data set WORK.TWO.
---
> NOTE: There were 4 observations read from the data set WORK.TWO.
110c110
<       real time           0.14 seconds
---
>       real time           0.13 seconds
```

This is a small application, but a larger application can have dozens or even hundreds of page header records, CPU/real time records, or other compare blocks that aren't of interest, making log comparisons more difficult. That was the impetus for writing the DIFFSPLIT macro to split diff output into two files.

MACRO DIFFSPLIT: SYNTAX

Macro DIFFSPLIT is called as follows.

%DIFFSPLIT(file1=*logfile1*,file2=*logfile2*,outfile=*outfileprefix*);

Required arguments:

file1=logfile1
First log file to compare. The file must exist.

file2=logfile2
Second log file to compare. The file must exist.

outfile=outfileprefix
File name for the two result files, which are named *outfileprefix*.useful.txt and *outfileprefix*.useless.txt.

MACRO DIFFSPLIT: NOTES

1. The macro looks for the start of a compare block that contains either a single pair of records or multiple records to see if they meet one of the "useless" criteria. Since we need access to multiple records at once, all records are copied into a temporary array at the start of the macro.

2. A few simplifying assumptions in the macro are as follows.
- diff begins the first record or group of records that differ with < and the second record or groups of records that differ with >. To simplify the regular expressions, we assume that < and > are ordered properly and just match either character by testing as follows: [\<\>]. To be robust, additional testing could be added.
- Elements of the temporary character array that holds all the records have a length of 200, which assumes the longest record is 200 characters. If necessary, the temporary array can be defined with a larger element size in part 4 of the code below.

3. Here are some examples of the first record of common compare blocks generated by diff.

The first record of a compare block that might be "useless" has 1 of 2 forms:

A single record compare.
```
361c361          (record numbers match)
361c365          (record numbers differ)
```

A multi record compare.
```
   785,786c785,786        (record numbers match)
   3885,3890c3890,3895    (record numbers differ)
```

Here are some other common examples of the first record of a compare block.
```
   23c34,36               (1 record to multiple records)
   42a58,62               (42 in file 1 and 57 in file 2 are the same,
                          extra records 58-62 in file 2)
   44,76c64,126           (blocks are different size, record numbers)
   167,223d248            (166 in file 1 and 223 in file 2 are the same,
                          extra records 167-223 in file 2)
```

MACRO DIFFSPLIT: CODE

Each section of the macro is described below.

PART 1. DEFINE THE MACRO AND DO SOME SIMPLE PARAMETER ERROR CHECKING

```
   %macro diffsplit(file1=,file2=,outfile=);

     %if "&file1" = "" %then %do;
        %put ERROR: First file to compare not entered, macro terminates;
        %return;
     %end;
     %else %if "&file2" = "" %then %do;
        %put ERROR: Second file to compare not entered, macro terminates;
        %return;
     %end;
     %else %if "&outfile" = "" %then %do;
        %put ERROR: Name for output files not entered, macro terminates;
        %return;
     %end;
     %else %if %sysfunc(fileexist("&file1")) ne 1 %then %do;
        %put ERROR: First file to compare (&file1) does not exist, macro
terminates;
        %return;
     %end;
     %else %if %sysfunc(fileexist("&file2")) ne 1 %then %do;
        %put ERROR: Second file to compare (file2) does not exist, macro
terminates;
        %return;
     %end;
```

PART 2. CREATE "DIFF" FILE WITH DIFFERENCES BETWEEN THE 2 FILES

The diff file name includes the date and time to ensure a unique name so the macro can be run multiple times without overwriting the output.

- General filename format: /tmp/diffsplit_yyyymmdd_hhmmss
- Example: /tmp/diffsplit_20210810_165308 for a file created on 8/10/2021, 4:53:08pm

```
      /* Create macro variables: current DATE as yyyymmdd, current time as hhmmss */
     data _null_;
       call symput ('datecurrent',put (date(),yymmddn8.));
       call symput ('timecurrent',compress(put(time(),time.),":"));
     run;
```

```
        /* File with differences */
      %let diffile=/tmp/diffsplit_&datecurrent._&timecurrent;

      x "diff &file1 &file2 > &diffile";
      %if %sysfunc(fileexist("&diffile")) ne 1 %then %do;
         %put ERROR: Unable to create "diff" file with difference between files, macro
terminates;
         %return;
      %end;
```

PART 3. DETERMINE NUMBER OF RECORDS IN THE DIFF FILE

- The number of records is used to create a temporary array containing the contents of the diff file in the next section.
- The Linux command wc -l counts the number of diff file records. We use a pipe to read the results into SAS. Comparable code to count the number of records in Windows is as follows:
  filename pipe1 pipe "find /c /v """" < &diffile";

```
filename pipe1 pipe "wc -l < &diffile";
data _null_;
  infile pipe1 ;
  input num_recs;
  call symputx("num_recs",num_recs);
run;
%put number of records in diff file is: &num_recs;
%if &num_recs = 0 %then %do;
 x "rm &diffile";
 %put ERROR: "Diff" file is empty, macro terminates;
 %return;
%end;
```

PART 4. READ THE DIFF FILE INTO A TEMPORARY CHARACTER ARRAY

- Each temporary array element has a length of 200, which assumes the longest record is 200 characters. This can be increased if necessary.
- Use TRUNCOVER because the record length varies and the strings have imbedded blanks.
- Delete the diff file after reading it.

```
data _null_;
  infile "&diffile" truncover;
  array all_records (&num_recs) $200 _temporary_;
  do i=1 to &num_recs;
    input all_records(i) $200.;
  end;
  call system ("rm &diffile");
```

PART 5. CREATE TWO FILES FOR DIFF RESULTS: A "USELESS" FILE AND A "USEFUL" FILE

```
  filename useless "&outfile..useless.txt";
  filename useful "&outfile..useful.txt";
```

PART 6. CREATE REGULAR EXPRESSIONS USED TO PARSE THE DIFF FILE RECORDS

```
    /* Complete record: numbers then c then numbers,
       for example 394c394 */
  regex1=prxparse("/^[0-9]+c[0-9]+\s*$/");
```

```
         /* Complete record: numbers then comma then numbers then c then
            numbers then comma then numbers, for example 414,415c414,415.
            Capture all 4 numbers.  */
         regex2=prxparse("/^([0-9]+),([0-9]+)c([0-9]+),([0-9]+)\s*$/");

         /* Complete record: < or > then space(s) then real time or cpu time.
            Example:    for regex3: <       real time           0.02 seconds
                        for regex4: <       cpu time            0.01 seconds
         */
        regex3=prxparse("/^[\<\>]\s+real time/");
        regex4=prxparse("/^[\<\>]\s+cpu time/");

          /* Complete record: < or > then space(s) then optionally
             (for pages 2 and beyond) ^L then page-number then space(s)
             then The SAS System, for example
             <  ^L7                             The SAS System 10:40
Saturday, November 7, 2020
          */
        regex5=prxparse("/^[\<\>]\s+\^?L?[0-9]+\s+The SAS System/");
```

PART 7. LOOP THROUGH THE TEMPORARY CHARACTER ARRAY WITH THE DIFF RECORDS, WRITE RECORDS TO FILES

Blocks with CPU/real time differences or page headers are written to the useless file, and everything else is written to the useful file.

```
        current_rec = 1; /* start from beginning of array */
        do while (current_rec lt &num_recs); /* not at end of array */

          if prxmatch(regex1,all_records(current_rec)) then do;
            /* Since regex1 matches a record of the form 394c394
               we know there are 4 records in the current block
               (the current record + 3 more) */

              /* Check for blocks with real time or cpu time */
            if (prxmatch(regex3,all_records(current_rec+1)) and
                prxmatch(regex3,all_records(current_rec+3))) or
               (prxmatch(regex4,all_records(current_rec+1)) and
                prxmatch(regex4,all_records(current_rec+3))) then do;
                file useless mod;
                do i=0 to 3;
                  put all_records(current_rec+i);
                end;
            end;

              /* Check for Page headers with The SAS System */
            else if (prxmatch(regex5,all_records(current_rec+1)) and
                prxmatch(regex5,all_records(current_rec+3))) then do;
                file useless mod;
                do i=0 to 3;
                  put all_records(current_rec+i);
                end;
            end;

              /* no more useless blocks w/4 records, must be a useful block */
            else do;
```

```
           file useful mod;
            do i=0 to 3;
              put all_records(current_rec+i);
            end;
        end;

        current_rec=current_rec+4; /* bypass current block */
      end; /* of if prxmatch(regex1,all_records(current_rec)) then do; */

      else if prxmatch(regex2,all_records(current_rec)) then do;
        /* Since regex2 matches a record of the form 46,47c46,47
           the number of records in the current block is
           ((2nd number - 1st number)+1) + ((4th number - 3rd number)+1)+2
           Extract the 4 numbers and calculate the number of records.
           buffer1=1st value, buffer2=2nd value from start of string etc.
        */
        buffer1 = input(trim(prxposn(regex2, 1, all_records(current_rec))),4.);
        buffer2 = input(trim(prxposn(regex2, 2, all_records(current_rec))),4.);
        buffer3 = input(trim(prxposn(regex2, 3, all_records(current_rec))),4.);
        buffer4 = input(trim(prxposn(regex2, 4, all_records(current_rec))),4.);
     /* Simplify blocksize=buffer2-buffer1+buffer4-buffer3+1+2; a bit */
        blocksize=buffer2-buffer1+buffer4-buffer3+4;

          /* real time / cpu time */
        if  blocksize=6 and (buffer2-buffer1=buffer4-buffer3) and
            prxmatch(regex3,all_records(current_rec+1)) and
            prxmatch(regex3,all_records(current_rec+4)) and
            prxmatch(regex4,all_records(current_rec+2)) and
            prxmatch(regex4,all_records(current_rec+5)) then do;
            file useless mod;
            do i=0 to 5;
              put all_records(current_rec+i);
            end;
        end;

        else do; /* useful block */
           file useful mod;
            do i=0 to blocksize-1;
              put all_records(current_rec+i);
            end;
        end;

        current_rec=current_rec+blocksize; /* bypass current block */
      end; /* of if prxmatch(regex1,all_records(current_rec)) then do; */

      else do; /* a record we are not checking for, just write it out */
         file useful mod;
         put all_records(current_rec);
         current_rec=current_rec+1; /* bypass current record */
      end;

    end; /* of do while (current_rec lt num_recs); */
    stop;
  run;
%mend diffsplit;
```

INVOKE DIFFSPLIT: EXAMPLE

```
%diffsplit (file1=/my/directory/path/sastests/alltest.20201107.log,
           file2=/my/directory/path/sastests/alltest.20201108.log,
           outfile=/my/directory/path/sastests/testdiff);
```

Results are written to the following files:

- /my/directory/path/sastests/testdiff/useless.txt
- /my/directory/path/sastests/testdiff/useful.txt

PORT DIFFSPLIT TO PLATFORMS OTHER THAN LINUX

To port macro DIFFSPLIT to another platform, the primary issue is finding a system command that generates output comparable to the Linux diff command. I could not find a standard Windows command or utility that generated results identical to diff. The closest analog I found was the fc command with the /L, /N, and /1 options, as in the following command.

```
     fc /L /N /1 log1.txt log2.txt > logdifftwindows.txt
```

- /L compares files as ASCII text.
- /N displays line numbers during an ASCII comparison.
- /*nnnn* specifies the number of consecutive lines that must match after a mismatch. Default is 2. 1 is used above.
- log1.txt and log2.txt are the two log files to compare.
- logdifftwindows.txt contains the result of the command.

Diff displays just lines that differ, but fc also shows the line before and after each line or block of lines that differ. Some recoding would be required to account for the extra lines.

The code to determine the number of records in the file with differences differs by platform. On Linux, this statement in Part 3 of DIFFSPLIT specifies a pipe that reads into SAS the result of a Linux command to determine the number of records in the diff file.

```
     filename pipe1 pipe "wc -l < &diffile";
```

The code can easily be generalized to test if it's running on (for example) Linux or Windows and use an appropriate system command with code like the following. Note that Windows requires two double quotes so they are doubled up in the code.

```
     %if %substr(&sysscp,1,2) eq WI %then %do; /* Windows */
       filename pipe1 pipe "find /c /v """" < &diffile";
     %end;
     %else %do; /* Linux */
       filename pipe1 pipe "wc -l < &diffile";
     %end;
```

EXTEND DIFFSPLIT FOR ADDITIONAL USELESS TEXT: A SIMPLE EXAMPLE

In the code above, only a few types of text were written to the "useless" file. It is easy to extend this to other cases. In general, the following two steps are required.

- Define a regular expression that describes the layout of the record(s) in question.
- Add conditional logic to test for the regular expression and when found, write the compare block to the useless file.

Let's illustrate this with a simple example.

My test script had many DATA steps where a FILE statement specified where text was written, as (very simply) illustrated in the following code.

```
data one;
  file '/my/directory/path/file1.txt';
  put "hello world";
run;
```

This generated log results like the following.

```
NOTE: The file '/my/directory/path/file1.txt' is:
      Filename=/my/directory/path/file1.txt,
      Owner Name=m1xxx00,Group Name=gg,
      Access Permission=-rw-rw-r--,
      Last Modified=23Sep2021:18:06:14
```

The diff file contained dozens of compare blocks like the following for files that had different date/time information.

```
4953c4953
<       Last Modified=23Sep2021:18:06:14
---
>       Last Modified=24Sep2021:18:06:29
```

To move these compare blocks to the "useless" file, modify the DIFFSPLIT macro as follows.

1. Add this statement to the end of Part 6 to define an additional regular expression. The end of the regular expression includes ",*" because the record sometimes includes a trailing comma.

```
    regex6=prxparse("/^[\<\>]\s+Last Modified=[0-9][0-9][a-zA-Z]{3}?[0-
9]{4}?:[0-9][0-9]:[0-9][0-9]:[0-9][0-9],*\s*$/");
```

2. Add the following statements in Part 7 after the 7 records of code that follow this comment: Check for Page headers with The SAS System.

```
    /* Check for records of form: <       Last Modified=19May2021:11:57:28 */
    else if (prxmatch(regex6,all_records(current_rec+1)) and
      prxmatch(regex6,all_records(current_rec+3))) then do;
      file useless mod;
      do i=0 to 3;
        put all_records(current_rec+i);
      end;
    end;
```

CONCLUSION

SAS log files from two executions of the same program are frequently compared. For example, we might compare today's results to a baseline run that we know is correct or test if a change to the SAS release, operating system, external data, our code, or anything else has altered the results.

Log file comparisons done by eyeballing the results of the Linux diff command can be difficult for programs with many steps because the diff output is often cluttered with many useless differences such as step summaries with real and/or CPU time differences and page headings with date/time differences.

The DIFFSPLIT macro runs diff for two SAS log files, parses the diff output with Perl Regular Expressions in a DATA Step, and splits the diff output into two files: one with useful file differences, and the other with useless differences, allowing us to focus on just the differences of interest.

The macro in this paper treated a few basic cases (real and/or CPU time differences and page headings with date/time differences) on Linux as useless, but as shown in the paper, it can easily be extended to run on other platforms or split the text using different criteria.

REFERENCES

fc command, Microsoft documentation. Accessed February 9, 2022. https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/fc.

find command, Microsoft documentation. Accessed February 9, 2022. https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/find.

diff(1) — Linux manual page. Accessed February 9, 2022. https://man7.org/linux/man-pages/man1/diff.1.html.

ACKNOWLEDGMENTS

Support from the following people is greatly appreciated. Ross Bettinger, Sean Dunn, and Zoriana Kurzeja all contributed substantially to the development of this paper.

CONTACT INFORMATION

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

Bruce Gilsen
Federal Reserve Board, Mail Stop N-122, Washington, DC 20551
202-452-2494
bruce.gilsen@frb.gov

APPENDIX: THE DIFFSPLIT MACRO

```
%macro diffsplit(file1=,file2=,outfile=);

/* PART 1. Define the macro and do some simple parameter error checking */
     %if "&file1" = "" %then %do;
        %put ERROR: First file to compare not entered, macro terminates;
        %return;
     %end;
     %else %if "&file2" = "" %then %do;
        %put ERROR: Second file to compare not entered, macro terminates;
        %return;
     %end;
     %else %if "&outfile" = "" %then %do;
        %put ERROR: Name for output files not entered, macro terminates;
        %return;
     %end;
     %else %if %sysfunc(fileexist("&file1")) ne 1 %then %do;
       %put ERROR: First file to compare (&file1) does not exist, macro terminates;
        %return;
     %end;
     %else %if %sysfunc(fileexist("&file2")) ne 1 %then %do;
       %put ERROR: Second file to compare (file2) does not exist, macro terminates;
        %return;
     %end;

/*  PART 2. Create "diff" file with differences between the 2 files.
    The diff file name includes the date and time to ensure a unique name
    and allow multiple executions of the macro.
    General filename format: /tmp/diffsplit_yyyymmdd_hhmmss
    Example: /tmp/diffsplit_20210810_165308 for a file created on 8/10/2021, 4:53:08pm
*/
      /* Create macro variables: current DATE as yyyymmdd, current time as hhmmss */
     data _null_;
       call symput ('datecurrent',put (date(),yymmddn8.));
       call symput ('timecurrent',compress(put(time(),time.),":"));
     run;
       /* File with differences */
     %let diffile=/tmp/diffsplit_&datecurrent._&timecurrent;

     x "diff &file1 &file2 > &diffile";
     %if %sysfunc(fileexist("&diffile")) ne 1 %then %do;
        %put ERROR: Unable to create "diff" file with difference between
files, macro terminates;
        %return;
     %end;

/*  PART 3. Determine number of records in the diff file.
    The number of records is used to create a temporary array containing the
    contents of the diff file in the next section.
    The Linux command wc -l counts the number of diff file records.  We use a
    pipe to read the results into SAS.  Comparable code to count the number
    of records in Windows is as follows:
         filename pipe1 pipe "find /c /v """" < &diffile";
*/
     filename pipe1 pipe "wc -l < &diffile";
     data _null_;
       infile pipe1 ;
       input num_recs;
       call symputx("num_recs",num_recs);
     run;
     %put number of records in diff file is: &num_recs;
     %if &num_recs = 0 %then %do;
      x "rm &diffile";
      %put ERROR: "Diff" file is empty, macro terminates;
      %return;
     %end;

/*  PART 4. Determine number of records in the diff file.
      Read the diff file into a temporary character array.
      Each temporary array element has a length of 200, which assumes the
      longest record is 200 characters.  This can be increased if needed.
      Use TRUNCOVER because the record length varies and the strings have
      blanks.
      Delete the diff file after reading it.
*/
       data _null_;
       infile "&diffile" truncover;
       array all_records (&num_recs) $200 _temporary_;
       do i=1 to &num_recs;
         input all_records(i) $200.;
       end;
       call system ("rm &diffile");

/*  PART 5. Create two files for diff results: a "useless" file and a
       "useful" file. */
     filename useless "&outfile..useless.txt";
     filename useful "&outfile..useful.txt";

/*  PART 6. Create regular expressions used to parse the diff file records.*/
         /* Complete record: numbers then c then numbers,
            for example 394c394 */
       regex1=prxparse("/^[0-9]+c[0-9]+\s*$/");

        /* Complete record: numbers then comma then numbers then c then
           numbers then comma then numbers, for example 414,415c414,415.
           Capture all 4 numbers.  */
        regex2=prxparse("/^([0-9]+),([0-9]+)c([0-9]+),([0-9]+)\s*$/");

        /* Complete record: < or > then space(s) then real time or cpu time.
           Example:    for regex3: <       real time           0.02 seconds
                       for regex4: <       cpu time            0.01 seconds
        */
       regex3=prxparse("/^[\<\>]\s+real time/");
       regex4=prxparse("/^[\<\>]\s+cpu time/");

         /* Complete record: < or > then space(s) then optionally
            (for pages 2 and beyond) ^L then page-number then space(s)
            then The SAS System, for example
            <  ^L7                              The SAS System 10:40
Saturday, November 7, 2020
         */
       regex5=prxparse("/^[\<\>]\s+\^?L?[0-9]+\s+The SAS System/");

/*  PART 7. Loop through the temporary character array with the diff records.
            Blocks with cpu/real time differences or page headers are written
            to the useless file, and everything else is written to the useful
            file.
*/
       current_rec = 1; /* start from beginning of array */
       do while (current_rec lt &num_recs); /* not at end of array */

         if prxmatch(regex1,all_records(current_rec)) then do;
           /* Since regex1 matches a record of the form 394c394
              we know there are 4 records in the current block
              (the current record + 3 more) */

             /* Check for blocks with real time or cpu time */
           if (prxmatch(regex3,all_records(current_rec+1)) and
               prxmatch(regex3,all_records(current_rec+3))) or
              (prxmatch(regex4,all_records(current_rec+1)) and
               prxmatch(regex4,all_records(current_rec+3))) then do;
               file useless mod;
               do i=0 to 3;
                 put all_records(current_rec+i);
               end;
           end;

             /* Check for Page headers with The SAS System */
           else if (prxmatch(regex5,all_records(current_rec+1)) and
               prxmatch(regex5,all_records(current_rec+3))) then do;
               file useless mod;
               do i=0 to 3;
                 put all_records(current_rec+i);
               end;
           end;

             /* no more useless blocks w/4 records, must be a useful block */
           else do;
              file useful mod;
               do i=0 to 3;
                 put all_records(current_rec+i);
               end;
           end;

           current_rec=current_rec+4; /* bypass current block */
         end; /* of if prxmatch(regex1,all_records(current_rec)) then do; */

         else if prxmatch(regex2,all_records(current_rec)) then do;
           /* Since regex2 matches a record of the form 46,47c46,47
              the number of records in the current block is
              ((2nd number - 1st number)+1) + ((4th number - 3rd number)+1)+2
              Extract the 4 numbers and calculate the number of records.
              buffer1=1st value, buffer2=2nd value from start of string etc.
           */
           buffer1 = input(trim(prxposn(regex2, 1, all_records(current_rec))),4.);
           buffer2 = input(trim(prxposn(regex2, 2, all_records(current_rec))),4.);
           buffer3 = input(trim(prxposn(regex2, 3, all_records(current_rec))),4.);
           buffer4 = input(trim(prxposn(regex2, 4, all_records(current_rec))),4.);
        /* Simplify blocksize=buffer2-buffer1+1+buffer4-buffer3+1+2; a bit */
           blocksize=buffer2-buffer1+buffer4-buffer3+4;

             /* real time / cpu time */
           if  blocksize=6 and (buffer2-buffer1=buffer4-buffer3) and
               prxmatch(regex3,all_records(current_rec+1)) and
               prxmatch(regex3,all_records(current_rec+4)) and
               prxmatch(regex4,all_records(current_rec+2)) and
               prxmatch(regex4,all_records(current_rec+5)) then do;
               file useless mod;
               do i=0 to 5;
                 put all_records(current_rec+i);
               end;
           end;

           else do; /* useful block */
              file useful mod;
               do i=0 to blocksize-1;
                 put all_records(current_rec+i);
               end;
           end;

           current_rec=current_rec+blocksize; /* bypass current block */
         end; /* of if prxmatch(regex1,all_records(current_rec)) then do; */

         else do; /* a record we are not checking for, just write it out */
            file useful mod;
            put all_records(current_rec);
            current_rec=current_rec+1; /* bypass current record */
         end;
```

```
    end; /* of do while (current_rec lt num_recs); */
    stop;
  run;
%mend diffsplit;
```