Paper 102-26

Merging Small Data sets To Large Unsorted Tape Data sets without a Sort Step, Using SAS Formats

William E Benjamin Jr, Phoenix, AZ

ABSTRACT

At times external flags or data need to be added to large unsorted tape files. The data may come from different systems or time periods, but need to be added to be able to represent relationships not available alone. Tape files with several million records and nearly a thousand variables take a long time to sort, if they can be sorted. The SAS FORMAT Procedure can be used to build a SAS data FORMAT to find a sparse set of records that require updating. The data can be added to the tape file records, using a SAS Data step to find affected records without sorting the tape file. This is, of course, slower than a merge but it can usually be faster than a sort. In this way small amounts of data can be added to the tape data set. This is a process available in Base SAS to all levels of skill.

INTRODUCTION

Often large unsorted tape files pose problems. They are big, single user, sequential, slow, and at times difficult to sort. Large files strain system limits and challenge programmers to become more creative than the system. This technique of merging data sets avoids some of the problems associated with large tape data sets.

BACKGROUND

Many systems still use tape files to archive data for long periods. Some legacy systems can output daily files with millions of large records. Using these files after they have been converted to SAS files with hundreds of variables can be expensive and time consuming.

Sorting files consume system resources in addition to the records on the file. Most sorts read all file records into memory, or onto disk reserves, as a part of the sort process before being able to rearrange the records, changing the order for the sorted output file. When the size of the tape file exceeds the amount of available disk resources for the sort routine the file becomes unsortable. Additionally, some files approaching the system limit sizes can be sorted, but require long sort execution times. Some can exceed several hours of system resources.

PROBLEM

There are many times that small amounts of data need to be merged onto records on large unsorted tape files, or a limited number of records need to be extracted from the file. One case is when time related data is created. For instance, in an inventory application, a storage bin location may change. Another case is when records are on a list the records need to be extracted for further processing and data needs to be added to the tape or updated. But, if the record is not on the list the new fields need to be added with default or null values. Most programmers are accustomed to merging sorted files. SAS software makes this process relatively simple, FOR SORTED FILES! But, large unsorted tape files may take several hours to read once, without including the time to sort the file and rewrite the tape.

SOLUTION

A simple one-pass method is needed to read and update the unsorted file. When a small file, or list of records needs to be processed against the large tape file, it is possible to be able to use two features of Base SAS, called formats and functions, to apply updates to the tape file. Each record from the tape file is checked against records on the format and processed accordingly. SAS formats, when used with a SAS ‘PUT’ function, provide a way to convert the value of a variable into a string of characters available to the Data step. This is a useful feature because the value returned can be tested and processed in many ways, and the value of the original variable and its format are not changed. This provides a way determine whether or not to apply data from the disk file or apply default values.

STEP ONE – CREATE A FORMAT

A format can be created for a small file or list, and can be created in any of several methods using the SAS FORMAT Procedure. I like the methods described below for small lists and files.
One of the drawbacks of using files with millions of records is that every thing your program does is repeated millions of times. Therefore, you must be careful to use the fewest number of steps and the fastest types of commands, or the job costs begin to soar. The object of using SAS formats is to place the list of records (format and data) into memory and search for matches using the speed of the format’s binary search to retrieve records from memory. A format can be constructed such that a match will return a value, and if no match is found an alternate default value will be returned, as in the following example:

```
    Proc Format ;
        Value $onlist
        '0001' = '35'
        '0003' = '43'
        '0006' = '10'
        other  =  'NO';
    Run;
```

A second method would be the following:

```
       Data Newfmt;

        Fmtname = '$onlist';
        Start   = 'OTHER';
        Label   = 'NO';
        OUTPUT;
        Start   = '0001';
        Label   = '35';
        OUTPUT;
        Start   = '0003';
        Label   = '43';
        OUTPUT;
        Start   = '0006';
        Label   = '10';
        OUTPUT;

Run;

Proc Format cntlin=newfmt;
```

```
Run;
```

The Format created by both sets of preceding code has a four character value and a two character label. In the following example the four character value is a part number, and the two character value is a bin number. The data in the value list is not required to be in sorted order. The following information is available, (but not in this layout), by using the FMTLIB option on with the Proc Format statement, along with the SAS Version and Date and time the format was created.

```
FORMAT NAME:            $ONLIST
LENGTH:                 2
NUMBER OF VALUES:       4
MIN LENGTH:             1
MAX LENGTH:             40
DEFAULT LENGTH          2
FUZZ:                   0

START           END             LABEL
0001            0001            35
0003            0003            43
0006            0006            10
**OTHER**       **OTHER**       NO
```

TEAS DATA FILE

Tth following is code to create a small disk file to illustrate the technique. This process is most effective on the large unsortable tape files, but will work on any file.

```
Data partfile ;

      Infile cards;

      Input @01      Part_no            $char4.
            @11      bin_no             $char2.
            @18      price              5.2
            @29      desc               $char30.
      ;
*Part_no      Bin_no      Cost      Description;
*2345678901234567890123456789012345678901234567 8
90;
cards;
0001      15      5.43      Hammer
0012      17      7.17      Screw Driver no2
0003      23      8.12      Screw Driver no4
0004      42      9.43      Screw Driver no9
0005      15      7.43      Rip Saw
0006      36      2.50      Flashlight
0017      93      6.75      Lantern no3
0008      47      8.31      Shovel
0009      53      9.77      Rake
0010      60      5.33      Pick
0011      16      8.78      Axe
0012      33      9.89      Drop Cloth
;;;;
Run;
proc print ;
```

```
OBS  PART_NO  BIN_NO    PRICE    DESC
1    0001      15        5.43     Hammer
2    0002      17        7.17     Screw Driver no2
3    0003      23        8.12     Screw Driver no4
4    0004      42        9.43     Screw Driver no9
5    0005      15        7.43     Rip Saw
6    0006      36        2.50     Flashlight
7    0017      93        6.75     Lantern no3
8    0008      47        8.31     Shovel
9    0009      53        9.77     Rake
10   0010      60        5.33     Pick
11   0011      16        8.78     Axe
12   0012      33        9.89     Drop Cloth
```

The file created by the preceding code is a file called “Partfile**”**. The file has no index and has three variables. But, it could have several hundred variables. In this case, the records are not in sorted order by the variable Part_no.

SLOW SOLUTION

Note that the left side of the value elements above can be part numbers or customer numbers or any record identifying information. These values are values that you may expect to find on the tape file, and may occur more than one time or in any order, on the tape file. The right side of the value clause is the data that you are going to use in the update process. This data may be actual values to add to the output file or codes to trigger internal processing or formatting. Here you can get creative. Base SAS formats allow an ‘OTHER’ clause for the case when the value being tested is not a valid value on the format. In a process such as this, you are expecting the large percentage of records to NOT be a valid value, and should code accordingly. The following code will work to merge the values from the format with the data listed at the end of the paper.

```
Data Merged;
    Set Partfile;
    chgbin = put (Part_No, $onlist.);
    If (chgbin ne 'NO') then do;
         Bin_No = chgbin;
         output  Merged;
         end;
    If (chgbin eq 'NO') then output Merged;
      Drop chgbin;
      Run;


proc print ;

run ;
```

```
OBS  PART_NO  BIN_NO    PRICE    DESC
1    0001      35        5.43     Hammer
2    0012      17        7.17     Screw Driver no2
3    0003      43        8.12     Screw Driver no4
4    0004      42        9.43     Screw Driver no9
5    0005      15        7.43     Rip Saw
6    0006      10        2.50     Flashlight
7    0017      93        6.75     Lantern no3
8    0008      47        8.31     Shovel
9    0009      53        9.77     Rake
10   0010      60        5.33     Pick
11   0011      16        8.78     Axe
12   0012      33        9.89     Drop Cloth
```

NOTES

As Noted above this will work, but not very fast, for the following reasons:

1. Assignment of a value, to the variable chgbin, for every record on the file is a waste of time, for the records which do not need to be changed. Especially since this kind of a test should only be used for merging very small files with very large files.
2. The code above has two “if” statements, which each test the value of the `chgbin`. Since the test is either to match a value or not match then only an ‘If….then….Else’ construct is required to test the values.
3. Within the ‘If … Then … Else’ the test case that will trigger most should be first , i.e. after the ‘Then’ clause, to speed processing.
4. Every instruction has system overhead associated with the

instruction. So by combining tasks within fewer instructions the system overhead can be reduced. Even a small reduction can add up when done millions of times.

5. By combining the 'if' condition test and the 'put' function the program will run faster because the assignment of a value to the variable 'chgbin' only occurs when the variable is needed. It is anticipated that the following code will run much faster even with a second 'Put' function when matches are found:
6. Since the output of a 'Put' function is ALWAYS a string, other string functions can be used to separate the data returned from the format into values that can be used within the data step.

FASTER SOLUTION

Note that the following code incorporates the changes above to enhance the program and speed of execution.

```
Data Merged2;
      Set Partfile;
      If  (put (Part_no,$onlist.) EQ 'NO')
       Then Do;
          Output Merged2;
       End;
       Else Do;
          Bin_no = put (Part_no, $onlist.);
         Output Merged2;
       End;
   Run;


proc print ;

run;

OBS  PART_NO  BIN_NO    PRICE    DESC
1   0001      35       5.43    Hammer
2   0012      17       7.17    Screw Driver no2
3   0003      43       8.12    Screw Driver no4
4   0004      42       9.43    Screw Driver no9
5   0005      15       7.43    Rip Saw
6   0006      10       2.50    Flashlight
7   0017      93       6.75    Lantern no3
8   0008      47       8.31    Shovel
9   0009      53       9.77    Rake
10  0010      60       5.33    Pick
11  0011      16       8.78    Axe
12  0012      33       9.89    Drop Cloth
```

While the placement of, (or need for) the 'Output' statements within DO … END blocks, are for illustrative purposes, this construct allows other code such as setting a default value for new variables.

CONCLUSION

The main idea is that the procedure described here will merge data to a file when it is not, or cannot, be sorted. The use of SAS Formats can be a useful tool that will allow the resourceful programmer to process an unsorted tape file and be able to merge data to the file.

REFERENCES

SAS Institute Inc. (1990), SAS Language: Reference, Version 6, First Edition, Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1990), SAS Procedures Guide, Version 6, Third Edition, Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1997), SAS Macro Language: Reference, First Edition, Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1991), SAS Technical Report P-222, Changes and Enhancements to Base SAS Software, Release 6.07, Cary, NC: SAS Institute Inc.

CONTACT INFORMATION

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

William E Benjamin Jr
1531 W Michigan Ave
Phoenix AZ, 85023
E-mail: WmEBenjaminJr3@juno.com