

Splitting Files: Use of the FILEVAR option for external files, and a programmed macro approach for SAS® data sets

Delayne Stokke, GuideOne Insurance, West Des Moines, IA

ABSTRACT

In my programming experience, I've often had a need to split data into multiple files based on a field that is present in the input file. The FILEVAR option of the FILE statement provides the required functionality for splitting data into multiple external files. However, I've found that there is quite a bit of confusion about how to code and use this option. The first part of this paper will explain how to use this option, and will highlight some common problems that may be encountered.

The second part of this paper will explain a macro method that can be used to easily split a file into multiple SAS® data sets, again based on a field that is present in the input.

INTRODUCTION

Suppose you've been asked to split a file of customer data by State, so that you end up with one external file for every state represented in the input. What's your first step? Audit the file to determine what states are represented? Then write some data steps to subset the data? What happens next month when new customers, possibly in new states are added to the file? Do you start over at the beginning?

This situation needs an automated way to create the required output files, without having to be concerned with which states are represented in the input file. Fortunately, the SAS System provides us with a way to get this job done.

PART 1: USE OF THE FILEVAR OPTION TO CREATE MULTIPLE EXTERNAL FILES

FILEVAR is an option on the FILE statement that defines a variable whose change in value causes the FILE statement to close the current output file and open a new one. The next PUT statement that executes writes to the new file specified as the value of the FILEVAR=variable. The FILEVAR can exist in the input file, or can be built, "on-the-fly" from data in the file. For optimum performance, the input should be grouped by the FILEVAR.

Consider the following input data set that needs to be split by the variable STATE:

OBS	NAME	STATE	PRODUCT
1	Jim	IA	Gizmos
2	Bill	KS	Whatzits
3	Jane	MN	Thingamajigs
4	Ralph	IA	Whatzits
5	Fred	KS	Gizmos
6	Julie	KS	Gizmos
7	Barney	MN	Thingamajigs
8	Chris	IA	Thingamajigs

FIRST THINGS FIRST – THE FILENAME STATEMENT

In order to use the FILEVAR option, you must code a FILENAME statement. There are a couple of interesting things about this requirement. The file specified on the FILENAME statement...

- ... is never opened by the program.
- ... and doesn't even need to exist!

You can assign the FILEREF to an existing file or to a "dummy" file, and your program will operate exactly the same way. You'll reference the FILEREF on the FILE statement in your DATA step, but when you code the FILEVAR option, SAS will ignore the file you specified on the FILENAME statement and use the file found in the FILEVAR variable instead.

Example FILENAME statement

```
filename my_file  
'c:\Demo\Files\my_file.txt';
```

ASSOCIATING AN OUTPUT FILE NAME WITH EACH RECORD

Perhaps the next thing that the programmer needs to consider is the creation of the output file names. In this example, I've built a variable named "Out_name". For each record processed, this new variable contains the output destination as a complete path and file name. The value of "Out_name" will change based on the value of the input variable "state".

Example assignment statement

```
Out_name =  
'c:\Demo\Files\'||state||'.txt';
```

For our example, the result is that:

- Iowa records will be directed to c:\Demo\Files\IA.txt
- Kansas records will be directed to c:\Demo\Files\KS.txt
- Minnesota records will be directed to c:\Demo\Files\MN.txt.

Note that if the input file contained records for an additional state, the output file specification would be created for that state as well.

CODING THE FILE STATEMENT WITH THE FILEVAR OPTION

To code the FILE statement, we'll start just as if we were going to direct all of the output to the file that we named on our FILENAME statement, above. However, we'll include the FILEVAR option, coded as "filevar=Out_name". In the following example, the file associated with the FILEREF "my_file" will be ignored and SAS will use the file found in the variable "Out_name" instead.

Example FILE statement

```
file my_file  
      filevar=Out_name;
```

Refer to the complete program named "Split into multiple External Datasets v1.1.SAS" in figure 1. Here's a look at the output files created after running this version of the program:

➤ C:\Demo\Files\IA.TXT contains:

Jim	IA	Gizmos
Ralph	IA	Whatzits
Chris	IA	Thingamajigs

➤ C:\Demo\Files\KS.TXT contains:

Bill	KS	Whatzits
Fred	KS	Gizmos
Julie	KS	Gizmos

➤ C:\Demo\Files\MN.TXT contains:

Jane	MN	Thingamajigs
Barney	MN	Thingamajigs

IF YOUR INPUT DATA ISN'T SORTED (OR GROUPED), YOUR RESULTS MAY VARY...

The previous version of the program included a PROC SORT step to make sure that the data were sorted on the split variable. But, the DATA step will run even if the data are not sorted. Review the input data set and refer to the program named "Split into multiple External Datasets v1.2.SAS" in figure 2. This is the same program but without the SORT step, and it results in the following output files:

➤ C:\Demo\Files\IA.TXT now contains:

Chris	IA	Thingamajigs
-------	----	--------------

➤ C:\Demo\Files\KS.TXT now contains:

Fred	KS	Gizmos
Julie	KS	Gizmos

➤ C:\Demo\Files\MN.TXT now contains:

Barney	MN	Thingamajigs
--------	----	--------------

So, what happened? Every time the input state changes, the corresponding output is opened as a "new" file. This causes any previously written data to be lost when the file is opened for the 2nd (or 3rd, 4th, etc.) time.

To prevent losing any data, you might be tempted to code a MOD option on your FILE statement to allow SAS to append records to an existing file. Refer to the program named "Split into multiple External Datasets v1.3.SAS" in figure 3. This is the same as the previous program, but has had the MOD option added to the FILE statement.

Example FILE statement

```
file my_file  
      filevar=Out_name  
      mod;
```

I like to offer the following caution when discussing the MOD option: "The use of MOD may result in inefficient coding, unexpected results, or both!"

After running this program, let's take another look at our output files, and see if we understand the reason for the caution...

➤ C:\Demo\Files\IA.TXT now contains:

Chris	IA	Thingamajigs
Jim	IA	Gizmos
Ralph	IA	Whatzits
Chris	IA	Thingamajigs

➤ C:\Demo\Files\KS.TXT now contains:

Fred	KS	Gizmos
Julie	KS	Gizmos
Bill	KS	Whatzits
Fred	KS	Gizmos
Julie	KS	Gizmos

➤ C:\Demo\Files\MN.TXT now contains:

Barney	MN	Thingamajigs
Jane	MN	Thingamajigs
Barney	MN	Thingamajigs

Another problem! We now have duplicate records in our output files. This happened because our latest version of the program appended the data to the output created by the previous version of the program. This illustrates one of the potential hazards of using the MOD option.

CONCLUSION: PART 1

The FILEVAR option provides an easy way to split files based on variable values found in the file.

Although it's not apparent with a small file, splitting an unsorted file with the FILEVAR option could result in an extremely inefficient program. Every time the split variable changes, SAS has to close one file and open another. If this happens frequently as a result of unsorted data, your program will require a lot of resources just to handle the I/O processing.

Because of the potential for problems such as losing data, duplicating data, and excess I/O, I strongly recommend that you make sure your input data is grouped by the split variable, and that you only use the MOD option with caution.

PART 2: A PROGRAMMED MACRO APPROACH TO CREATE MULTIPLE SAS DATA SETS

Again, suppose you have the same problem as in Part 1 – you need to split a customer file into multiple outputs based on State – only this time, your output needs to be SAS data sets rather than external files. We'll use the same input data as in the previous example.

A simple example is offered here to suggest how we might create an output SAS data set for a single state:

```
data saswork.IA;  
  set demo;  
  where state = "IA";  
run;
```

Now, if we re-write this as a rather simple macro, we'll start to get closer to our solution:

```
%macro splitsas(infile,
                outfile,
                splitvar,
                splitval);

    data &outfile.;
        set &infile;
        where &splitvar = "&splitval";
    run;

%mend;

%splitsas(demo,saswork.IA,state,IA)
```

For a small number of states, you could write the Macro Call statements yourself. But what if...

- ... there are a lot of states to split?
- ...you run the job at different times and there are different states represented at different times?
- ...you are concerned that you may mis-code a split variable or output file name?

You can automate the creation of the Macro Call Statements! To do this, start by sorting the input by the split variable.

```
/* Sort Input file by Split Variable */
proc sort data=saswork.demo out=demo;
    by state;
run;
```

Next, create the Macro Call statements by processing one record for each unique split variable. The following program will create Macro Call statements for every state in the input file. The Call statements are created as a character variable named "callmacr", and are written to the external file "splitsas".

```
filename splitsas c:\Pgms\Splitsas.SAS';

/* Write Macro Call Statements to
external file */
data _null_;
    set demo;
    by state;

    /* Process One Record per State */
    if first.state;

    /* Create the Macro Call Statement */
    callmacr = '%splitsas('||
        'demo,'||
        'saswork.'||state||','||
        'state,'||
        state||
        ')';

    /* Identify the Output External File */
    file splitsas;

    /* Write the Macro Call Statement(s) */
    put callmacr;

run;
```

If we examine the contents of the external file "splitsas", here's what we'll see:

```
%splitsas(demo,saswork.IA,state,IA)
%splitsas(demo,saswork.KS,state,KS)
%splitsas(demo,saswork.MN,state,MN)
```

Each Macro Call statement that we need has been successfully created by SAS. Now all that's left to do is to read these statements back into SAS, using a "%include" statement:

```
/* Split the file */
%include splitsas;
```

Refer to the complete program named "Split into Multiple SAS Data Sets.SAS" in figure 4. This program creates the following output data sets:

- SASWORK.IA contains 3 observations:

OBS	NAME	STATE	PRODUCT
1	Jim	IA	Gizmos
2	Ralph	IA	Whatzits
3	Chris	IA	Thingamajigs

- SASWORK.KS contains 3 observations:

OBS	NAME	STATE	PRODUCT
1	Bill	KS	Whatzits
2	Fred	KS	Gizmos
3	Julie	KS	Gizmos

- SASWORK.MN contains 2 observations:

OBS	NAME	STATE	PRODUCT
1	Jane	MN	Thingamajigs
2	Barney	MN	Thingamajigs

CONCLUSION: PART 2

To create multiple SAS data sets, I've offered a programmed macro approach. The benefits of this approach are that it minimizes the potential for error and it maximizes the program's flexibility by allowing you to split ANY SAS data set based on ANY variable in the file.

The method used here requires that the input file be read every time the macro is called. For relatively small files the increased I/O processing is of little concern. If, however, your input file is relatively large you may want to explore another method to split your file.

REFERENCES

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

CONTACT INFORMATION

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

Delayne Stokke
GuideOne Insurance
1111 Ashworth Road
West Des Moines, IA 50265-3538
Work Phone: (515) 267-2212
Email: dstokke@guidemail.com or
dstokke@aol.com

```

/*      Program Name:   Split into multiple External Datasets v1.1.SAS
      Purpose:         Demo
      Date:            October 5, 1998
      Programmer:      Delayne Stokke
*/

```

```
libname saswork 'c:\Demo\Saswork';
```

```

/* NOTE: The file named in the following fileref statement
      will never be opened by this program.  The fileref
      is needed simply to satisfy the requirements of
      the "file" statement in the data step.          */

```

```
filename my_file 'c:\Demo\Files\my_file.txt';          This file doesn't need to exist, and won't be created by this program.
```

```
/* I recommend that the file to be split is sorted by the split variable */
```

```
proc sort data=saswork.demo out=demo;
```

```
  by state;          File must be sorted, or the data must be grouped (ie: all of Kansas is together, but occurs before Iowa)
run;
```

```

data _null_;
  set demo;
  length Out_name $ 200;

```

```
/* Build the output file name for each record */
```

```
Out_name = 'c:\Demo\Files\'||state||'.txt';          Build the output file name, which includes the "split" variable.
```

```
file my_file          /* Fileref "my_file"...  */
```

Fileref "my_file" is needed just to satisfy the requirements of the "file" statement.

```
  filevar=Out_name;  /* ...is over-ridden by the file
                    identified by the "filevar" option. */
```

The actual output file is named by the variable "Out_name".

```

put name      $char20. +1
  state      $char2.  +1
  product    $char20.;

```

```
run;
```

Figure 1. - Split into multiple External Datasets v1.1.SAS

```
/*      Program Name:   Split into multiple External Datasets v1.2.SAS
      Purpose:         Demo...
                        If your data is not sorted, results may vary.
      Date:            October 5, 1998
      Programmer:      Delayne Stokke
*/
```

```
libname saswork 'c:\Demo\Saswork';
```

```
/* NOTE: The file named in the following fileref statement
      will never be opened by this program.  The fileref
      is needed simply to satisfy the requirements of
      the "file" statement in the data step.          */
```

```
filename my_file 'c:\Demo\Files\my_file.txt';
```

```
data _null_;
  set saswork.demo;
```

Input file is not sorted or grouped.

```
length Out_name $ 200;
```

```
/* Build the output file name for each record */
Out_name = 'c:\Demo\Files\'||state||'.txt';
```

```
file my_file          /* Fileref "my_file"... */
```

```
filevar=Out_name; /* ...is over-ridden by the file
                  identified by the "filevar" option. */
```

```
put name    $char20. +1
    state   $char2.  +1
    product $char20.;
```

```
run;
```

WARNING:

Output files are opened as "new" files every time the variable "Out_name" changes. If a file is opened 2 or more times, all previously written data are LOST!

Figure 2. - Split into multiple External Datasets v1.2.SAS

```
/*      Program Name:  Split into multiple External Datasets v1.3.SAS
      Purpose:        Demo...
```

```
          "mod" option can be handy in splitting a file that's
          not sorted.  But it may be inefficient if your data is
          not somewhat grouped together.
```

```
          Also, be careful if the files named by "filevar"
          already exist.  New data from this step will
          be appended to what's already there.
```

```
          Date:        October 5, 1998
          Programmer:   Delayne Stokke
```

```
*/
libname saswork 'c:\Demo\Saswork';
filename my_file 'c:\Demo\Files\my_file.txt';
```

```
data _null_;
  set saswork.demo;
```

```
  length Out_name $ 200;
```

```
  Out_name = 'c:\Demo\Files\'||state||'.txt';
```

```
  file my_file          /* Fileref "my_file"... */
```

```
    filevar=Out_name    /* ...is over-ridden by the file
                        identified by the "filevar" option. */
```

```
  mod;                  /* Caution: The use of "mod" may
                        result in inefficient coding,
                        unexpected results, or both! */
```

```
  put name      $char20. +1
      state     $char2.  +1
      product   $char20.;
```

```
run;
```

```
In most cases, I recommend NOT using this option.
```

Figure 3. - Split into multiple External Datasets v1.3.SAS

```
/*      Program Name:   Split into multiple SAS Data Sets.SAS
      Purpose:         Demo
      Date:            October 5, 1998
      Programmer:      Delayne Stokke
*/
```

```
libname saswork 'c:\Demo\Saswork';
filename splitsas 'c:\Demo\Programs\Splitsas.SAS';
```

```
/* Macro Definition */                                     General Purpose SAS Dataset Split Macro.
%macro splitsas(infile,outfile,splitvar,splitval);

  data &outfile;
    set &infile;
    where &splitvar = "&splitval";
  run;

%mend;
```

```
/* Sort Input file by Split Variable */                   File must be sorted.
proc sort data=saswork.demo out=demo;
  by state;
run;
```

...program is continued on the next page.

Figure 4. - Split into multiple SAS Data Sets.SAS

```

/* Write Macro Call Statements to external file */
data _null_;
  set demo;
  by state;
  /* Write one Macro Call Statement per State */
  if first.state;

  callmacr = '%splitsas('||
              'demo,'||
              'saswork.'||state||','||
              'state,'||
              state||
              ')';

  file splitsas;
  put callmacr;

run;

/* Split the file */
%include splitsas;

```

“by” statement and subsetting “if” needed to create one Macro Call Statement for each State in the input file.

Create the variable “callmacr” for each State.

/* Input SAS Dataset Name */

/* Output SAS Dataset Name */

/* Split Variable Name */

/* Split Variable Value */

Output file is the external fileref “splitsas”.

Write the Macro Call Statement to the Output file.

Macro Call statements in the fileref “splitsas” are “%included” in the program.

Here are the contents of the fileref “splitsas”.

```

%splitsas(demo,saswork.IA,state,IA)
%splitsas(demo,saswork.KS,state,KS)
%splitsas(demo,saswork.MN,state,MN)

```

Figure 4. - Split into multiple SAS Data Sets.SAS, continued