Paper 82-27

Using the FILEVAR= Option for Input and Output

Derek Morgan, Washington University Medical School, St. Louis, MO

Abstract

The problem of handling multiple flat or ASCII files with the SAS® System has been one that has been solved many times with macros. However, there is an easier way, one that saves resources, time, and programmer frustration. The FILEVAR= option on the infile statement takes care of switching among multiple ASCII files, and it works for both input and output. Working from production examples, this short paper will explain how to use it, and the advantages over traditional macro methods of handling this problem.

Old-time SAS System programmers like me used to use macros to handle multiple files for input or output. We'd read one file, create a SAS System data table, append it to the master template, and process the next file in the same fashion. The file name information would be stored in a macro variable, and the macro variable would change with each iteration of the loop. While this does have its place, you are no longer forced to do it this way.

The ***FILEVAR=column name*** option for the INFILE statement takes care of multiple files. The fully qualified name of each file to be processed is stored in ***column-name***. This can either be done in a previous DATA step, where one record for each file to be read exists in a separate SAS System data table, or it can be done "on the fly."

Using the FILEVAR= Option

How do you get the list of files in the first place? Under UNIX and Windows, you can use the redirection symbol ">" to take the output from a directory command and put it into a file. The format of the directory command for UNIX is:

ls *directory-and-file specifier* > *filename*

Directory-and-file specifier represents the files you wish to process and optionally, the directory where they are located. ***Filename*** indicates the name of the ASCII file where you want to send this listing. Therefore, if you wanted a list of all the files in the test1 subdirectory sent to a file named "project.filelist", the command would be:

ls /test1/* > project.filelist

Getting Path Names in UNIX

```
ls /path991006/data*/*.data > tmpit
```

This will list the names of all the files in the path991006 directory, where the subdirectory name begins with "data". If there are two subdirectories named "dataA", and "dataB", the file listing will have the following format:

```
/path991006/dataA/chr1.data
/path991006/dataA/chr2.data
/path991006/dataA/chr3.data
/path991006/dataB/chr1.data
/path991006/dataB/chr2.data
```

In Windows, the command is:

dir /b file specifier > filename

File-specifier represents the file or subdirectory you wish to process, and ***filename*** indicates the name of the ASCII file where you want to send this listing. One important item to remember about the redirection symbol (>) is that the file that is to receive the output cannot already exist. If this is part of a production job, you can always erase it before the job runs and the directory command is executed. Under Windows, you will not get the directory path as part of the file names. Therefore, you will have to modify the file name in the data table before you use it so that you have the complete file identifier in the data table.

As an example, let's take one of our actual production jobs. Here's one of the data files that will compose the data table. This data is obtained from a blood pressure machine that produces an ASCII file like this for each person whose blood pressure is measured.

```
FBPP PROTOCOL VERSION 10/16/2000
Date: 06/04/01
BP Certification ID: 038
Subject ID: A9174
TS  1 09:13.46 TIMESTAMP
LS  1 09:14.50 BBA07563100075615075066095058
LS  2 09:16.52 BBA07582300075822077072094058
RS  3 09:20.39 BBA08022000080209082068100063
RS  4 09:22.31 BBA08041200080401075066101061
TU  1 09:26.06 TIMESTAMP
LU  1 09:32.11 BBA08135200081341073054096057
TP  1 09:32.39 TIMESTAMP
LP  1 09:33.16 BBA08144800081448085065110063
TM  1 09:37.19 TIMESTAMP
LM  1 09:41.08 BBA08224800082237077063092060
LM  2 09:42.25 BBA08235500082355070063093055
TM  2 09:44.36 TIMESTAMP
LM  3 09:45.17 BBA08265700082647083068094066
LM  4 09:46.18 BBA08275900082748081069097063
TG  1 09:49.41 TIMESTAMP
LG  1 09:53.38 BBA08350900083509076068097054
LG  2 09:54.52 BBA08362300083623075062097056
TG  2 09:55.40 TIMESTAMP
LG  3 09:57.19 BBA08390000083844078065105056
TG  3 09:57.27 TIMESTAMP
```

On average, there are about 50 of these files in a batch. Each file has a different name, and the specific file names change from run to run of this job. This is how you do it to avoid coding any file names or changing the program each run.

Example 1: "On-the-fly"

```
1   x 'del bpfiles.lst';
2   x 'dir /b d:\cait\data\multifile\*.bp > bpfiles.lst';
3   DATA temp;
4   LENGTH bpfile $ 80 bpid $ 7;
5   INFILE 'bpfiles.lst' PAD MISSOVER;
6   INPUT bpfile $ 1-30;
7   bpfile = "d:\cait\data\multifile\" || TRIM(LEFT(bpfile));
8   RETAIN seq 0 bpid date;
9   INFILE in FILEVAR=bpfile END=done PAD
    MISSOVER;
10  DO UNTIL(done);          ①
11    INPUT arm $ 1 tst $ 2 @;
12     SELECT(arm);
13       WHEN('L','R') DO;
14         INPUT count 5-6 hours 7-8 mins 10-11
           secs 13-14 pulse 36-38 systolic 39-41
           diastolic 42-44;
15         time = HMS(hours,mins,secs);
16         IF time LT 0 THEN
17           TIME = .m;
18         OUTPUT;
19       END;
20       WHEN('T') DO;
21         INPUT count 5-6 hours 7-8 mins 10-11
           secs 13-14;
22         tstamp = HMS(hours,mins,secs);
23         OUTPUT;
24       END;
25       WHEN('S') INPUT bpid $ 13-19;
26       WHEN('B') INPUT techid $ 22-29;
27       WHEN('D') INPUT @7 date mmddyy8.;
28       OTHERWISE INPUT;
29     END;
30  END;
31  RUN;
```

This is an example of how to read many files on the fly with a single DATA step using the FILEVAR= option. The first two lines prepare the file listing for use in the program. Line 2 creates the file list. Lines 5-7 read this file. In line 7, the path name is prepended to the file name to yield a fully qualified filename. This value is stored in the column BPFILE. Line 9 is the INFILE statement. The word "in" is used as a dummy filename because a filename is required when you use the FILEVAR= option. However, it can be any legal SAS System filename that is not already assigned. The FILEVAR= option uses the column BPFILE to let the SAS System know which file to read. The END= option on the INFILE statement must be present when using FILEVAR=, for reasons which will be explained below. It is critical that you use this specific option.

Lines 10-30 (①) compose the DO UNTIL loop that processes each data file containing the blood pressure data. It is keyed to the end-of-file indicator that was set up with the END= option in the INFILE statement on line 9. Why do we need a loop to read the data in the files? Doesn't the DATA STEP automatically process a file until it reaches the end? Yes, but the DATA step loop is reading the raw data file pertaining to the INFILE statement on line 5 (which is the listing of the data files, not the actual blood pressure data files themselves.) Therefore, the DATA step will iterate once for each line in <u>that</u> file. In order to read all the raw data from the blood pressure files, it is necessary to loop through each blood pressure file. Without the loop, only the first record from each blood pressure file would be processed.

The OUTPUT statements (lines 18 and 23) are in the loop for a similar reason. Without them, only the last record from each blood pressure file would be output, since the default output operation would occur after the INPUT statement in line 6 was executed. In our example here, some of the data read from each blood pressure file will not be in the data table created, because there is no OUTPUT statement associated with the INPUT statement. The END= option tells the SAS System when to stop processing each blood pressure file (at end-of-file.) Below is an excerpt of the log created by the program above.

```
   (code removed)
NOTE: The infile 'bpfiles.lst' is:
      File Name=C:\bpfiles.lst,
      RECFM=V,LRECL=256

NOTE: The infile IN is:
      File Name=d:\cait\data\multifile\16028AF.BP,
      RECFM=V,LRECL=256

… (more notes about the files being read) …

NOTE: The infile IN is:
      File Name=d:\cait\data\multifile\46053CF.BP,
      RECFM=V,LRECL=256

NOTE: 50 records were read from the infile 'bpfiles.lst'.
      The minimum record length was 10.
      The maximum record length was 10.
NOTE: 28 records were read from the infile IN.
      The minimum record length was 14.
      The maximum record length was 44.

… (more notes about how many lines were read from
each file) …

NOTE: 28 records were read from the infile IN.
      The minimum record length was 14.
      The maximum record length was 44.
NOTE: The data set WORK.TEMP has 1188 observations
and 15 columns.
NOTE: DATA statement used:
      real time           0.22 seconds
```

A Second Method

The previous example may be a little confusing at first glance because of the way we are doing everything "on-the-fly." It is also possible to place the file listing into a SAS System table, and do the multiple input file processing from there. This is helpful if you have pairs or triples of files to process. It is slightly more explicit because you are not reading one file while trying to read many others. Let's take the same example, only instead of reading the file listing in the same DATA step, we'll read it into a SAS System data table first.

Example 2: From A Flat File

```
1   x 'del bpfiles.lst';
2   x 'dir /b d:\cait\data\multifile\*.bp > bpfiles.lst';
3   DATA filelisting;
4   LENGTH bpfile $ 80 bpid $ 7;
5   INFILE 'bpfiles.lst' PAD MISSOVER;
6   INPUT bpfile $ 1-30;
7   bpfile = "d:\cait\data\multifile\" || TRIM(LEFT(bpfile));
8   RUN;

9   DATA bpdata;
10  SET filelisting;
11  RETAIN seq 0 bpid date;
12  INFILE in FILEVAR=bpfile END=done PAD
    MISSOVER;
13  DO UNTIL(done);          ①
14    INPUT arm $ 1 tst $ 2 @;
15     SELECT(arm);
16       WHEN('L','R') DO;
17         INPUT count 5-6 hours 7-8 mins 10-11
           secs 13-14 pulse 36-38 systolic 39-41
           diastolic 42-44;
18         time = HMS(hours,mins,secs);
19         IF time LT 0 THEN
20           TIME = .m;
21         OUTPUT;
22       END;
23       WHEN('T') DO;
24         INPUT count 5-6 hours 7-8 mins 10-11
           secs 13-14;
25         tstamp = HMS(hours,mins,secs);
26         OUTPUT;
27       END;
28       WHEN('S') INPUT bpid $ 13-19;
29       WHEN('B') INPUT techid $ 22-29;
30       WHEN('D') INPUT @7 date mmddyy8.;
31       OTHERWISE INPUT;
32     END;
33  END;
34  RUN;
```

In the above example, we start with MS-DOS shell commands to delete the file listing if it exists, then create one using the proper directory command with the redirection operator.

The first DATA step (lines 3-8) reads in the file that was just created. Since this is MS-DOS, no path information is provided with the short file name listing, so we add the proper path in line 7 to give us a fully qualified file name. This is stored in the column BPFILE.

Instead of using an INPUT statement as we did in example 1, now we use the SET statement (line 10). Everything else is the same, including the use of the END= option in the INFILE statement. Again, the DATA step will execute once for each observation (input file) in the SET dataset. Therefore, the INPUT statement must be enclosed in a DO UNTIL loop in order to read more than the first record from the file name stored in the column BPFILE. Similarly, the OUTPUT statement must be used within this loop to create an observation for each record in each file. Otherwise, the dataset BPDATA will only have the last record from each of the input files being read.

Writing Data to Multiple Flat Files

This is a little easier; there are no DO loops to worry about. The dataset will need to have a column that indicates to which flat file the observation is to be written. This does not need to be a fully qualified file name in the dataset, but that will have to be done before it can be used as a FILEVAR.

In this example, let's say that you are producing flat files for 100 different departments, and each file should have the department name as a part of its file name. The column DEPT contains the department names.

```
1   PROC SORT DATA=tempb1;
2   BY dept date;
3   RUN;

4   OTPIONS NONOTES;
5   DATA _NULL_;
6   LENGTH fv $ 100;
7   SET tempb1;
8   fv = "/local4/" || TRIM(dept) || ".data";
9   FILE writeout FILEVAR=fv MOD;
10  PUT date date9. @10 type @30 source @50
    account z7. @57 amount dollar10.2;
11  RUN;
```

There are a few things to be aware of when doing this. The SAS System usually displays a message in the log when an external file is opened. If your data are not sorted so that each output file is only opened once, you will get a message each time the output file changes. In this example, you are writing to 100 different files, one record at a time. Without sorting the dataset by DEPT (lines 1-3), the log file would be enormous! Fortunately, if you forget to sort properly, the NONOTES option (line 4) will come to your rescue, and prevent the SAS log from filling a disk volume.

The rest of the writing operation is straightforward. In line 8, the external file name is assembled using the table column DEPT, and stored in the column FV. The FILE statement at line 9 must have a dummy filename (in this case, "writeout"), but the file is actually specified by the column in the FILEVAR option. The MOD option on the FILE statement prevents the file from being restarted each time it is opened. From there, it's just a matter of using whatever PUT statements you need to produce the desired output.

Summary

Here's a note of caution about running out and converting existing programs to this method. If you're responsible for maintaining legacy code that uses macros to read and/or write multiple files, unless you have time on your hands or it's causing a resource problem, it's probably best to leave it alone. However, this method of reading and writing multiple files uses one DATA step and no intermediate tables. It is a little more resource and programmer-time efficient than the other method.

This work was partially supported by NHLBI grants HL 54473 and HL 47317.

Further inquiries are welcome to:

Derek Morgan
Division of Biostatistics
Washington University Medical School
Box 8067, 660 S. Euclid Ave.
St. Louis, MO 63110
Phone: (314) 362-3685 FAX: (314) 362-2693
E-mail: derek@wubios.wustl.edu

This and other SAS System examples and papers can be found on the World Wide Web at:

http://www.biostat.wustl.edu/~derek/sasindex.html