

Using the PIPE and FILEVAR= Options

Richard D. Langston, SAS Institute Inc.

INTRODUCTION

This paper attempts to describe two very useful features of the INFILE and FILE statements within the SAS® System. These features are the FILEVAR= option and the PIPE option.

THE FILEVAR= OPTION

The FILEVAR= option allows you to specify a character variable whose value is used to determine the name of the file to be processed. Normally, the name of the file on an INFILE or FILE statement must be hard-coded. But since the INFILE and FILE statements are executable statements, the DATA step can examine the values of the FILEVAR= variable to determine the name of the file to open. If the file name is the same name as the last time that same INFILE/FILE statement was seen, no change occurs and the previously opened file will continue to be used. But if the file name differs from the last time the same INFILE/FILE statement was executed, the current file is closed and the newly named file is opened.

Any time the FILEVAR= filename changes, a log message reports this change. You may find that if you will be changing files a lot that you may want to use the NONOTES option to suppress these messages.

Note also that the triggering of changing files occurs only when the variable referenced by the FILEVAR= option changes value, and only in association with either the same INFILE statement or another INFILE statement with the same fileref.

Note also that the INFILE or FILE statement on which a FILEVAR= option must still have a fileref, although it is primarily used as a placeholder when the FILEVAR= option is used. The fileref value is important, however, if you have subsequent INFILE or FILE statements which are to inherit the same FILEVAR= attributes. Those subsequent statements must use the same fileref.

The FILEVAR= option is available on all Version 6 platforms.

THE PIPE OPTION

The PIPE option allows the filename on an INFILE or FILE statement to actually be a command that's invoked. For an INFILE statement, the command produces a data stream for its *stdout*, and this data stream becomes input via the INFILE statement. For a FILE statement, the command expects a data stream from its *stdin*, and the FILE statement passes data to the command's *stdin*.

The PIPE option can be used in conjunction with the FILEVAR= option, where the variable referenced in the FILEVAR= option can contain a command.

Note also that the PIPE option can be used on a FILENAME statement, but it must appear immediately after the fileref.

The PIPE option is available for all PC and UNIX platforms running Version 6 of the SAS System.

SAMPLE FILES

Our examples were developed for UNIX platforms, but they can

also apply to PC platforms (just use the backslash instead of the forward slash, and place temporary files in a special directory). Before examining our examples, let's suppose that we have a directory called *sampdir* with three files called *file1*, *file2*, and *file3*. Their contents are created as follows:

```
data _null_; file 'sampdir/file1';
  input; put _infile_; cards;
file1: line 1 of 3
file1: line 2 of 3
file1: line 3 of 3
run;
data _null_; file 'sampdir/file2';
  input; put _infile_; cards;
file2: line 1 of 2
file2: line 2 of 2
run;
data _null_; file 'sampdir/file3';
  input; put _infile_; cards;
file3: line 1 of 4
file3: line 2 of 4
file3: line 3 of 4
file3: line 4 of 4
run;
```

FIRST EXAMPLE - NO SPECIAL FEATURES

We'd like to create a SAS data set called ALLTEXT with two variables, FILE, containing the name a file, and LINE, containing the value of a line of data in the file. ALLTEXT should contain the contents of all files in the *sampdir* directory.

Without first considering the FILEVAR= or PIPE options, let's first see how we could produce such a SAS data set.

```
%macro readit(filename);
data temp;
  infile "&filename." length=1;
  length file $80;
  retain file "&filename.";
  input @; input @1 line $varying200. 1;
  run;
proc append base=alltext data=temp;
  run;
%mend;

x 'ls -l sampdir/* > /tmp/temp.dat';
data _null_;
  infile '/tmp/temp.dat' length=1;
  input @;
  input @1 filename $varying200. 1;
  file '/tmp/temp.sas';
  put '%readit(' filename $varying200. 1 ')';
  run;
%inc '/tmp/temp.sas'/source2;
run;
data _null_; set alltext; put _all_; run;
```

In the above code, we use the UNIX command *ls -l* to write the names of the files in *sampdir* into the temporary file called */tmp/temp.dat*. This UNIX command is invoked via the X command in the SAS language. Once we have the file list in */tmp/temp.dat*, we can run a DATA step to read each line of data to obtain the file names. We can then create another flat file called */tmp/temp.sas* that contains the macro invocation lines of the form

```
%readit(filename);
```

where *filename* will be the name of each file to be processed.

We also coded up the %readit macro that will read through the entire file, creating the FILE and LINE variables and appending the observations into ALLTEXT.

Once the macro invocation statements are in */tmp/temp.sas*, we can use the %INCLUDE statement to invoke the statements.

SECOND EXAMPLE - SOME IMPROVEMENT

The above example is rather clunky and involves the creation of several files. The next iteration avoids the creation of the */tmp/temp.sas* file, and instead uses the CALL EXECUTE statement.

```
proc delete data=alltext;
x 'ls -l sampdir/* > /tmp/temp.dat';
data _null_;
  infile '/tmp/temp.dat' length=1;
  input @;
  input @1 file $varying200. 1;
  call execute('%readit('||trim(file)||')');
  run;
data _null_; set alltext; put _all_; run;
```

THIRD EXAMPLE - USING THE PIPE OPTION

In this iteration, we avoid the creation of additional files. We can use a PIPE option and execute the *ls* command 'on the fly'.

```
proc delete data=alltext;
data _null_;
  infile 'ls -l sampdir/*' pipe length=1;
  input @;
  input @1 file $varying200. 1;
  call execute('%readit('||trim(file)||')');
  run;
data _null_; set alltext; put _all_; run;
```

FOURTH EXAMPLE - USING PIPE AND FILEVAR

In this iteration, we can avoid the creation of the macro and instead read through all the files in a single DATA step. We do this by using a second INFILE statement with the FILEVAR= option. We can then provide a DO WHILE loop to allow for all records in each file to be read.

```
proc delete data=alltext;
data alltext;
  infile 'ls -l sampdir/*' pipe length=1;
  input @;
  input @1 file $varying200. 1;
  filevar=file;
  infile xxx filevar=filevar end=eof length=12;
  do while(^eof);
    input @;
    input @1 line $varying200. 12;
    output;
  end;
  keep file line;
  run;
data _null_; set alltext; put _all_; run;
```

Note that the variable referenced in the FILEVAR= option is not

kept, regardless of whether it appears in a KEEP statement. Therefore, we ensure that we don't use the FILE variable in the FILEVAR= option since we want to keep that variable for later use.

USING THE FILEVAR AND PIPE OPTIONS TOGETHER

To demonstrate this feature, let us suppose that some or all of the files in *sampdir* were compressed. This means that the .Z extension would appear in the file name. This also means that we cannot INFILE directly from the individual files because they must first be uncompressed. What we can do here is to determine if the file is compressed and to create a command string of the form

```
zcat filename
```

for compressed files and

```
cat filename
```

for uncompressed files.

Both the *zcat* and *cat* write to *stdout*, allowing us to use the PIPE option on the INFILE statement. And since we want to produce the command for each different filename, we use a FILEVAR= option.

```
proc delete data=alltext;
data alltext;
  infile 'ls -l sampdir/*' pipe length=1;
  input @;
  input @1 file $varying200. 1;
  if substr(file,length(file)-1)='.Z'
    then command='zcat';
  else command='cat';
  filevar=trim(command)||' '||file;
  infile xxx pipe filevar=filevar
    end=eof length=12;
  do while(^eof);
    input @;
    input @1 line $varying200. 12;
    output;
  end;
  keep file line;
  run;
data _null_; set alltext; put _all_; run;
```

Note that if you are using the PIPE option on the INFILE or FILE statement, the PIPE keyword must appear immediately after the fileref.

ENSURING FILEVAR= VALUES CHANGE

Note that the triggering of closing and opening files via the FILEVAR= option occurs when the value of the FILEVAR= variable changes. Hitting end-of-file on a given file is not sufficient to trigger a closing.

Consider this example where we list a filename and want to display all lines associated with the file. We list *sampdir/file1* twice in succession:

```
options nonotes;
data _null_;
  infile cards length=1;
  input @; input @1 filevar $varying200. 1;
  infile xxx filevar=filevar end=eof;
  do while(^eof);
```

```

        input; put _infile_;
    end;
cards;
sampdir/file1
sampdir/file1
sampdir/file2
run;

```

We would expect to see two sets of the contents of *file1* followed by one set of *file2* contents on the log. Instead, we only see one set of each. This is because when we read the second line of CARDS data, we still have the FILEVAR variable set to 'sampdir/file1', so the INFILE statement does not detect a change. We will have already hit end-of-file for *sampdir/file1* (and the variable EOF will thus be set to 1), so we don't go through the DO WHILE loop. Only when FILEVAR is set to 'sampdir/file2' do we close *sampdir/file1* and continue.

To avoid this problem, use a dummy file and always open it after completing a run-through of the desired file:

```

data _null_;
    file '/tmp/dummy.dat';
    put 'x';
    run;

data _null_;
    infile cards length=1;
    input @; input @1 filevar $varying200. 1;
    infile xxx filevar=filevar end=eof;
    do while(^eof);
        input; put _infile_;
    end;
    filevar='/tmp/dummy.dat';
    infile xxx;
    return;
cards;
...

```

After the DO WHILE loop, we execute another INFILE statement. This statement inherits the attributes of the previous INFILE statement because the fileref XXX is the same. Since the variable FILEVAR has a different value, the file *sampdir/file1* is closed and */tmp/dummy.dat* is opened. (Note that the dummy file has to exist, even though we never read from it). When the next CARDS line is read, even though we go back to *sampdir/file1*, the filename has changed so we'll reopen it for processing.

CONCLUSION

The FILEVAR= option allows for dynamic file processing, which is a very powerful feature. The PIPE option allows for command execution and for avoiding the creation of temporary files. The two options together allow for efficient processing within single DATA steps.

REFERENCES

SAS is a registered trademark of SAS Institute Inc. in the USA and other countries.