Paper 9-26

MISSOVER, TRUNCOVER, and PAD, OH MY!!
or Making Sense of the INFILE and INPUT Statements.

Randall Cates, MPH, Technical Training Specialist

ABSTRACT

The SAS® System has many powerful tools to store, analyze and present data. However, first programmers need to get the data into SAS datasets. This presentation will delve into the intricacies of reading data from sequential (text) files using the DATA step and INFILE and INPUT statements. Discussion will focus on the different options available when reading different types of text files. For example, when should you use the MISSOVER option and when is the TRUNCOVER option more appropriate. This paper assumes the audience has basic knowledge of reading text files using the DATA step (Base SAS®) and is appropriate for users on any Operating System, although some options may be restricted.

INTRODUCTION

Reading and understanding the SAS documentation can sometimes be a challenge. This is evident in the INFILE statement. There are no less than 34 different options available for this particular statement. This can get very sticky when the data file you need to read differs from the safe, easy columnar data files. So how can we make sense of the plethora of options? This paper will attempt to clarify some of the confusion. Three situations are explored. First Variable-Length records; both shorter values, and missing data points. Next, reading in multiple files at once. Finally, obtaining data from both remote OS's and Web sites using the FILENAME statement.

SO LITTLE TIME, SO MANY OPTIONS

When the data lines aren't complete, what option will read the data correctly and completely? INFILE has a number of options available:

FLOWOVER	The default. Causes the INPUT statement to jump to the next record if it doesn't find values for all variables.
MISSOVER	Sets all empty vars to missing when reading a short line. However, it can also skip values.
STOPOVER	Stops the DATA step when it reads a short line.
TRUNCOVER	Forces the INPUT statement to stop reading when it gets to the end of a short line. This option will not skip information.
SCANOVER	Causes the INPUT statement to search the data lines for a character string specified in the INPUT.
PAD	Pads short lines with blanks to the length of the LRECL= option.

Note: SCANOVER and STOPOVER will not be discussed.

The following text file was created with MS-Notepad on Windows-NT then read into a SAS dataset using INFILE and INPUT statements. Each line should contain 4 data points; Last and First names, Employee ID and Job title. The grayed-out area denotes actual line lengths. (Note: Most Word processors on Windows and UNIX create variable-length lines, whereas Mainframe computers files with lines of uniform length, filled in by blanks.)

```
LANGKAMM        SARAH     E0045 Mechanic
TORRES          JAN       E0029 Pilot
SMITH           MICHAEL   E0065
LEISTNER        COLIN     E0116 Mechanic
TOMAS           HARALD
WADE            KIRSTEN   E0126 Pilot
WAUGH           TIM       E0204 Pilot
```

Then two sets of code were submitted using different options on the INFILE statement. First the lines were read in with Column Input;

```
DATA test;
    INFILE "d:\infile\emplist.dat"
           <OPTIONS>;
    INPUT lastn $1-21 Firstn $ 22-31
        Empid $32-36 Jobcode $37-45;
RUN;
```

Then List Input was used;

```
DATA test;
  INFILE "d:\infile\emplist2.dat";
  INPUT lastn $ Firstn $
          Empid $ Jobcode $ ;
RUN;
```

FLOWOVER:
The FLOWOVER option is the default option on INFILE. Here, when the INPUT statement reaches the end of non-blank characters without having filled all variables, a new line is read into the Input Buffer and INPUT attempts to fill the rest of the variables starting from column one. The next time an INPUT statement is executed, a new line is brought into the Input Buffer. The results (printed with PROC PRINT) are below.

Column Input;

Obs	Lastn	Firstn	Empid	Jobcode
1	LANGKAMM	SARAH	E0045	Mechanic
2	TORRES	JAN	E0029	SMITH
3	LEISTNER	COLIN	E0116	Mechanic
4	TOMAS	HARALD	WADE	WAUGH

In the second line, since the value PILOT did not extend to the required number of columns for Jobcode(37-45), the INPUT statement jumped to the next line to complete Jobcode. Similarly, for the fifth line read in, the INPUT statement first jumped to the sixth line to read Empid, then to the seventh line to read Jobcode.

List Input;

Obs	Lastn	Firstn	Empid	Jobcode
1	LANGKAMM	SARAH	E0045	Mechanic
2	TORRES	JAN	E0029	Pilot
3	SMITH	MICHAEL	E0065	LEISTNER
4	TOMAS	HARALD	WADE	KIRSTEN
5	WAUGH	TIM	E0204	Pilot

In this example the Pilot values are placed in the appropriate places, but the INPUT statement still loops to the next line when unable to fill all variables.

MISSOVER:
When the MISSOVER option is used on the INFILE statement, the INPUT statement does not jump to the next line when reading a short line. Instead, MISSOVER sets variables to missing.

Column input;

Obs	Lastn	Firstn	Empid	Jobcode
1	LANGKAMM	SARAH	E0045	Mechanic
2	TORRES	JAN	E0029	
3	SMITH	MICHAEL	E0065	
4	LEISTNER	COLIN	E0116	Mechanic
5	TOMAS	HARALD		
6	WADE	KIRSTEN	E0126	
7	WAUGH	TIM	E0204	

All lines are read in as separate records. Notice however, that the PILOT Jobcodes are still missing. When MISSOVER encounters the End-Of-Line mark, and has not read all required columns for a particular variable, then that variable is set to missing. This is better, but still not perfect.

List Input;

Obs	Lastn	Firstn	Empid	Jobcode
1	LANGKAMM	SARAH	E0045	Mechanic
2	TORRES	JAN	E0029	Pilot
3	SMITH	MICHAEL	E0065	
4	LEISTNER	COLIN	E0116	Mechanic
5	TOMAS	HARALD		
6	WADE	KIRSTEN	E0126	Pilot
7	WAUGH	TIM	E0204	Pilot

Since List Input doesn't specify explicit columns, these data lines can be correctly read using the MISSOVER option.

TRUNCOVER:
The TRUNCOVER option acts similarly to MISSOVER, and in addition, will take partial values to fill the first unfilled variable.

Column Input;

Obs	Lastn	Firstn	Empid	Jobcode
1	LANGKAMM	SARAH	E0045	Mechanic
2	TORRES	JAN	E0029	Pilot
3	SMITH	MICHAEL	E0065	
4	LEISTNER	COLIN	E0116	Mechanic
5	TOMAS	HARALD		
6	WADE	KIRSTEN	E0126	Pilot
7	WAUGH	TIM	E0204	Pilot

Here TRUNCOVER successfully reads the short lines, apportioning out the values to the correct places. When the INPUT statement reached a foreshortened line, the TRUNCOVER option takes what's left (e.g. Pilot) and assigns it to the appropriate value. Other variables are set to missing where necessary.

List Input;

Obs	Lastn	Firstn	Empid	Jobcode
1	LANGKAMM	SARAH	E0045	Mechanic
2	TORRES	JAN	E0029	Pilot
3	SMITH	MICHAEL	E0065	
4	LEISTNER	COLIN	E0116	Mechanic
5	TOMAS	HARALD		
6	WADE	KIRSTEN	E0126	Pilot
7	WAUGH	TIM	E0204	Pilot

Since List Input reads from delimiter to delimiter, TRUNCOVER can still work.

PAD:
The PAD option does not replace the FLOWOVER option. Instead, the PAD option adds blanks to short lines out to the logical record length(LRECL). In this case, PAD takes the LRECL from the file information, but you can specify LRECL= in the INFILE statement.

Column Input;

Obs	Lastn	Firstn	Empid	Jobcode
1	LANGKAMM	SARAH	E0045	Mechanic
2	TORRES	JAN	E0029	Pilot
3	SMITH	MICHAEL	E0065	
4	LEISTNER	COLIN	E0116	Mechanic
5	TOMAS	HARALD		
6	WADE	KIRSTEN	E0126	Pilot
7	WAUGH	TIM	E0204	Pilot

When reading in data with Column Input, SAS reads "just the columns, Ma'am". Since the PAD option adds blanks, SAS can read the appropriate columns without hitting the End-of-File mark. So the data is read in correctly.

List Input;

Obs	Lastn	Firstn	Empid	Jobcode
1	LANGKAMM	SARAH	E0045	Mechanic
2	TORRES	JAN	E0029	Pilot
3	SMITH	MICHAEL	E0065	LEISTNER
4	TOMAS	HARALD	WADE	KIRSTEN
5	WAUGH	TIM	E0204	Pilot

List Input reads data from delimiter to delimiter. The default delimiter character is a blank. Multiple delimiters are treated as one. So with the PAD option in effect, and FLOWOVER still in effect, the INPUT statement must look to the next line to fill the remaining variables.

SYNOPSIS:
Reading files with variable line lengths can be frustrating, especially when one doesn't fully understand how each option does, and doesn't, work. The default option of FLOWOVER expects to fill all variables, and uses multiple lines if necessary.

MISSOVER was originally created to be used in conjunction with PAD and works effectively and well in most situations. However, this can be a CPU intensive process when reading an extremely large file.

STOPOVER is a good tool for checking code and raw data when dealing with large, potentially messy files, since it forces the DATA step to stop the first time it finds a short line.

TRUNCOVER was developed later than the MISSOVER and PAD options, and deals admirably with not only short lines but with short values. TRUNCOVER is more also efficient since it doesn't require the extra "padding".

One more point about variable-length files. It is possible to copy in a subset of any raw data file into the DATA step and run these options on the subset. Use an INFILE DATALINES; statement, and add whichever options are appropriate. For example;

```
DATA test;
  INFILE datalines TRUNCOVER;
    INPUT lastn $1-20 firstn $21-30
        empid $31-35 jobcode $37-44;
DATALINES;
"add a number of data lines here sans
semicolons"
RUN;
```

ALL THE FILES, PLEASE
Another situation that might come up is where the raw data exists in numerous multiple files. Here the INFILE statement has a couple of options that can help. The FILEVAR= option allows us to specify a variable, to be filled during DATA step execution, that will contain the name of the raw data file. The END= option allows us to set a variable that registers, for each raw line read in, "is this the last line of the file?". We can use these in a number of useful ways to input data from multiple files to a SAS Dataset. Here are a few possibilities.

First, just list the files in a series of DATALINES in the DATA step.

```
DATA one;
  LENGTH fil2read $ 40;
  INPUT fil2read $;
  INFILE dummy FILEVAR=fil2read
        END=done;
  DO WHILE (NOT done);
    INPUT lastn $ firstn $
          hiredate : mmddyy8.
          salary;
    OUTPUT;
  END;
DATALINES;
D:\Infile\emplist.dat
D:\Infile\emplist1.dat
D:\Infile\emplist2.dat
D:\Infile\emplist3.dat
D:\Infile\emplist4.dat
RUN;
```

The first INPUT statement reads each line and saves the information to a temporary variable. Add an INFILE statement with FILEVAR= set to the variable just read in. Then set up an INPUT/OUTPUT loop to read each file.

Be careful to set up the DO loop so that the DATA step never gets to the End-Of-File marker on any file. Using the END= option on the second INFILE statement sets up a temporary variable (done) which will register 0 (not the last record) or 1 (the last record) for each raw data line read in from each file. This is necessary since, if SAS reads in any End-of-File marker, the DATA step closes. By testing for DONE at the top of the loop (DO WHILE), and exiting the DO loop after the last line of every file, we ensure that we never hit the end-of-file for all files read in. This remains true even for empty files.

A SAS Dataset can be used to store the names of the files and would be called using a SET statement.

```
DATA one;
  set two;
  INFILE dummy FILEVAR=fil2read
         END=done;
  DO WHILE (NOT done);
    INPUT lastn $ firstn $
          hiredate : mmddyy8.
          salary;
    OUTPUT;
  END;
RUN;
```

Finally, it's possible to read in filenames dynamically, using a FILENAME with the Pipe option. This is useful when all of the files are in the same directory. With the PIPE keyword, the FILENAME statement can take an operating system command in quotes, and accept the result as valid input. Unfortunately, this is not available on Mainframe operating systems.

```
FILENAME indata PIPE
         "dir D:\Infile\*.dat /b";
DATA test;
  LENGTH fil2read $40;
  INFILE indata MISSOVER;
  INPUT fil2read $;
  fil2read="d:\infile\"||fil2read;
  INFILE dummy FILEVAR=fil2read
        END=done;
  DO WHILE(NOT done);
    INPUT lastn $ firstn $
          hiredate : mmddyy8.
          salary;
    OUTPUT;
  END;
RUN;
```

The information returned from the FILENAME statement is a list of all files in D:\Infile with a .DAT type. One can specify all files, or (as above) specific files. The DATA step can use this information with one INFILE statement and then use the information to read the files by applying it to a FILEVAR= option on a second INFILE statement.

One limitation is that the Windows command (DIR) returns only the names of the files without the pathnames. So the fil2read variable needs to be augmented with the pathname in an assignment statement.

```
fil2read="d:\infile\"||fil2read;
```

In UNIX, a similar FILENAME statement would read:

```
FILENAME indata PIPE
   "ls -l /Infile/*.dat /b";
```

The UNIX **ls** command returns a fully qualified path and filename.

THE FILES ARE WHERE??

This last topic is a little off subject; i.e. you can use the FILENAME and FTP to access and read files on another operating system. The FILENAME also has a URL access method to read a file at a Web site. Once a data source has been defined

by the FILENAME statement, a DATA step is able to access, open and read the data using usual INFILE/INPUT statements.

To access remote files using the FILENAME FTP Access method, there are a number of options to tell SAS how to get to the data. Fortunately, if one is at all familiar with FTP the options are relatively straightforward.

This example prompts the user for a password, connects to a UNIX server, moves to a particular directory (/Infile/Mydata) reads a file named emplist.dat in the directory, and dumps each record into one variable in the output dataset test.

```
filename unix ftp 'emplist.dat'
        cd='Infile/Mydata'
        user='racate'
        host='test.unix.sas.com'
        prompt;
data test;
  length name $ 300;
  infile unix truncover;
  input name $;
run;
```

Other options are;

DEBUG	Writes information to the SAS log about the FTP process.
LRECL=	Logical record length of remote file.
PASS=	Password to use on remote server.
RECFM=	Record format. "F", "S", "V"

Accessing Web pages is similar to the above code. Define a connection to a Web page/site using FILENAME with the URL option, defining an http web site as the pathname, with other options as necessary, then use DATA step coding to read the file.

This example accesses a web page on the SAS Institute Inc.'s web site, reads the first 15 lines of html code, and writes them to the log.

```
filename foo url
'http://www.sas.com/service/techsup/intr
o.html';
data _null_;
   infile foo length=len;
   input record $varying200. len;
   put record $varying200. len;
   if _n_=15 the stop;
run;
```

CONCLUSION

This paper has described some options of the FILENAME statement for different situations. There are many different types of data files, and SAS can read in most, if not all. SAS can read data files of variable-lengths, delimited files, files with missing data, multiple files per DATA step, files on other operating systems, even HTML Web pages. With a broader knowledge of the SAS's data reading capabilities, programmers can accept data from multiple sources with confidence.

REFERENCES

SAS Institute Inc., SAS® Language Reference, Version 8, Cary, NC: SAS Institute Inc., 1999. 1256 pp.
SAS Institute Inc., SAS® Companion for the Microsoft Windows Environment, Version 8, Cary, NC: SAS Institute Inc., 1999. 555 pp.
SAS Institute Inc., Technical Support Notes, TS-581, Using FILEVAR= to read multiple external files in a DATA Step, http://ftp.sas.com/techsup/download/technote/ts581.html, Cary, NC: SAS Institute Inc., 2000. 5 pp.

CONTACT INFORMATION

Randall Cates, Technical Training Specialist II
SAS Institute Inc.
St. Louis Regional Office
MCI Building, Suite 550
100 South Fourth St.
St. Louis, MO 63102
(314)421-6364 ext. 8506
Randall.Cates@sas.com