Paper 166-2008

The SAS INFILE and FILE Statements

Steven First, Systems Seminar Consultants, Madison, WI

ABSTRACT

One of the most flexible features of the SAS system, is its ability to read and write just about any kind of raw file. The INFILE and FILE statements are the interfaces that connect SAS programs to those external files so that INPUT and PUT can read and write data. These statements provide many options to make reading and writing simple to complex files in an easy way.

INTRODUCTION

This paper will examine the INFILE and FILE statements and their options. To access a file from any programming language, a linkage is needed between the program and the desired file. INFILE and FILE are the statements that are used in SAS to generally link to *raw* files; that is, files that normally contain only data and no data dictionary. INFILE is used to point t*o input* files and FILE points to *output* files. In many ways, other than the direction of data flow, INFILE and FILE act the same and do have many of the same options. There are also many unique options for INFILE versus FILE.

Because there is normally no dictionary describing raw data, it is up to the program to provide enough information so that SAS can read/write the data in or out of the data step. INFILE/FILE statements have extensive options to help provide that information and allow SAS to process a rich and varied range of files with minimal effort. INFILE/FILE also work with other SAS statements such as FILENAME, DATALINES, PUT and INPUT to provide extensive data input and output in the DATA step. Because of the ability to read and write just about any kind of file format with SAS, we have a wonderfully versatile utility tool via the INFILE and FILE statements.

BASIC INFILE SYNTAX

INFILE *file-specification* <*options* > <*operating-environment-options*> ;

INFILE *DBMS-specifications;*

file-specification

identifies the source of the input data records, which is an external file or instream data. *File-specification* can have these forms:

'*external-file*'

specifies the physical name of an external file. The physical name is the name that the operating environment uses to access the file.

fileref

specifies the fileref of an external file. (note: The fileref must be previously associated with an external file in a FILENAME statement, FILENAME function, or a system command)

fileref(file)

specifies a fileref of an aggregate storage location and the name of a file or member, enclosed in parentheses, that resides in that location.

Operating Environment Information: Different operating environments call an aggregate grouping of files by different names, such as a directory, a MACLIB, or a partitioned data set. Details are given in SAS operating system documentation.

CARDS | CARDS4
DATALINES | DATALINES4

Note: Complete INFILE and FILE documentation are included as appendices at the end of this paper.

HOW TO USE THE INFILE/FILE STATEMENT

Because the INFILE statement identifies the file to read, it must execute before the INPUT statement that reads the input data records. The same holds true for the FILE statement which must precede any PUT statement that performs the writing to the output raw file.

```
data x;                          /* build SAS dataset */
  infile in;                     /* raw file in       */
  input @1  Name $10.            /* read a record     */
        @20 Age    2. ;          /* with two fields   */
run;                             /* end of step       */
```

READING MULTIPLE FILES

You may specify more than one INFILE statement to read more than one file in a single data step. The step will stop when any file tries to read past the end of file. You may also use the INFILE statement in conditional processing, such as an IF-THEN statement, because it is executable. This enables you to control the source of the input data records. It is a bit more difficult to read multiple flat files in a data step as compared to reading multiple SAS datasets. For that reason it is a bit unusual to read multiple files in one step.

The following DATA step reads from two input files during each iteration of the DATA step. As SAS switches from one file to the next, each file remains open. The input pointer remains in place to begin reading from that location the next time an INPUT statement reads from that file.

```
data qtrtot(drop=jansale febsale marsale   /* build a dataset and drop input  */
                 aprsale maysale junsale); /* variables                       */
   infile file-specification-1;            /* identify location of 1st file   */
   input name $ jansale febsale marsale;   /* read values from 1st file       */
   qtr1tot=sum(jansale,febsale,marsale);   /* sum them up                     */
   infile file-specification-2;            /* identify location of 2nd file   */
   input @7 aprsale maysale junsale;       /* read values from 2nd file       */
   qtr2tot=sum(aprsale,maysale,junsale);   /* sum them up                     */
run;                                       /* end of step                     */
```

The DATA step terminates when SAS reaches an end of file on the shortest input file.

A HEX LISTING PROGRAM

The INPUT statement does not need to specify any variables to read; it simply reads a line into a buffer from our input file. This can be useful with the LIST statement to just display the raw input file's buffer. LIST will display hexadecimal notation if unprintable characters are present in the file.

```
data _null_;                     /* don't need dataset*/
  infile in;                     /* raw file in       */
  input;                         /* read a record     */
  list;                          /* list buffer in log*/
  if _n_ > 50 then               /* stop after 50     */
     stop;                       /* adjust as needed  */
run;                             /* end of step       */
```

ACCESSING THE INPUT BUFFER

After an INFILE and INPUT statement executes, the buffer area can be accessed and even altered through a special variable called _INFILE_. This allows a simple way to read and alter data without extensive coding in the INPUT statement to define the records. Another way that the previous program could display the input buffer area is shown below:

```
data _null_;                    /* don't need dataset*/
  infile in;                    /* raw file in       */
  input;                        /* read a record     */
  put _infile_;                 /* put  buffer in log*/
  if _n_ > 50 then              /* stop after 50     */
     stop;                      /* adjust as needed  */
run;                            /* end of step       */
```

Not only can we display the buffer, we can actually alter the values before reading individual fields. Suppose we have the following file that contains angle brackets that we would like to discard.

```
City Number Minutes Charge
Jackson 415-555-2384 <25> <2.45>
Jefferson 813-555-2356 <15> <1.62>
Joliet 913-555-3223 <65> <10.32>
```

In the following code, the first INPUT statement reads and holds the record in the input buffer. The compress function removes the angle brackets (< >) from special field _INFILE_. The second INPUT statement parses the value in the buffer and then PUT displays the SAS variables. Note that the FIRSTOBS INFILE option skips the first header record.

```
data _null_;
   length city number $16. minutes charge 8;
   infile phonbill firstobs=2;
   input @;
   _infile_ = compress(_infile_, '<>');
   input city number minutes charge;
   put city= number= minutes= charge=;
run;
```

Partial SAS log:

```
city=Jackson number=415-555-2384 minutes=25 charge=2.45
city=Jefferson number=813-555-2356 minutes=15 charge=1.62
city=Joliet number=913-555-3223 minutes=65 charge=10.32
```

ASSIGNING ANOTHER VARIABLE TO THE CURRENT BUFFER

The _INFILE_=variable names a character variable that references the contents of the current input buffer for this INFILE statement. This variable like all automatic variables is not written to the SAS dataset. It may be useful to define this type of variable rather than use _INFILE_ especially when multiple files are being input. The results from the following program are identical to those of the above.

```
data _null_;
   length city number $16. minutes charge 8;
   infile phonbill firstobs=2 _infile_=phonebuff;
   input @;
   _infile_ = compress(phonebuff, '<>');
   input city number minutes charge;
   put city= number= minutes= charge=;
run;
```

READING INSTREAM DATA RECORDS WITH INFILE

You may use the INFILE statement with the DATALINES file specification to process instream data and still utilize other INFILE options. An INPUT statement reads the data records that follow the DATALINES statement. Again, the results from this next program match the earlier ones. Note that if the system option CARDIMAGE is on, the record is assumed to be an 80 byte record padded with blanks. For longer dataline input, OPTIONS NOCARDIMAGE may need to be specified.

```
data _null_;
   length city number $16. minutes charge 8;
   infile datalines firstobs=2 _infile_=phonebuff;
   input @;
   _infile_ = compress(phonebuff, '<>');
   input city number minutes charge;
   put city= number= minutes= charge=;
datalines;
City Number Minutes Charge
Jackson 415-555-2384 <25> <2.45>
Jefferson 813-555-2356 <15> <1.62>
Joliet 913-555-3223 <65> <10.32>
;
run;
```

READING PAST THE END OF A LINE

By default, if the INPUT statement tries to read past the end of the current input data record, it then moves the input pointer to column 1 of the next record to read the remaining values. This default behavior is governed by the FLOWOVER option and a message is written to the SAS log. This is useful in reading data that flows over into several lines of input.

Example: Read a file of pet names and 6 readings per animal.

```
data readings;
 infile datalines;
 input Name $ R1-R6;
 datalines;
Gus  22 44 55
     33 32 14
Gaia 24 22 23
     31 76 31
;
proc print data=readings;
 title 'Readings';
 run;
```

Partial SAS log:

```
NOTE: SAS went to a new line when INPUT statement reached past the end of a line.
```

SAS Output

Readings

Obs	Name	R1	R2	R3	R4	R5	R6
1	Gus	22	44	55	33	32	14
2	Gaia	24	22	23	31	76	31

CONTROLLING SHORT RECORDS WITH FORMATTED INPUT

With data that doesn't flow over, the FLOWOVER behavior can cause errors. Several options are available to change the INPUT statement behavior when an end of line is reached. The STOPOVER option treats this condition as an error and stops building the data set. The MISSOVER option sets the remaining INPUT statement variables to missing values. The SCANOVER option, used with @'character-string' scans the input record until it finds the specified *character-string*. The FLOWOVER option restores the default behavior.

The TRUNCOVER and MISSOVER options are similar. After passing the end of a record both options set the remaining INPUT statement variables to missing values. The MISSOVER option, however, causes the INPUT statement to set a value to missing if the statement is unable to read an entire field because the field length that is specified in the INPUT statement is too short. The TRUNCOVER option writes whatever characters are read to the last variable so that you know what the input data record contained.

For example, an external file with variable-length records contains these records:

```
----+----1----+----2
1
22
333
4444
55555
```

The following DATA step reads this data to create a SAS data set. Only one of the input records is as long as the informatted length of the variable TESTNUM.

```
data numbers;
   infile 'external-file';
   input testnum 5.;
run;
```

This DATA step creates the three observations from the five input records because by default the FLOWOVER option is used to read the input records. This output of course is not correct.

If you use the MISSOVER option in the INFILE statement, then the DATA step creates five observations. However, all the values that were read from records that were too short are set to missing.

The INFILE TRUNCOVER option tells INPUT to read as much as possible and that the value will be chopped off.

```
infile 'external-file' truncover;
```

The table below shows the results of the three different INFILE options.

The Value of TESTNUM Using Different INFILE Statement Options

OBS	FLOWOVER	MISSOVER	TRUNCOVER
1	22	.	1
2	4444	.	22
3	55555	.	333
4		.	4444
5		55555	55555

ANOTHER SOLUTION

The INFILE LRECL= and PAD options can be used to specify a record width and if the actual line is shorter to fill (pad) it with blanks. The program below reads the data correctly as all records are assumed to be 5 wide.

```
data numbers;
   infile 'external-file' lrecl=5 pad;
   input testnum 5.;
run;
proc print data=numbers;
title 'Numbers';
run;
```

Numbers

Obs	testnum
1	1
2	22
3	333
4	4444
5	55555

HANDLING SHORT RECORDS AND MISSING VALUES WITH LIST INPUT

The example below shows data records that sometimes don't contain all six readings. With FLOWOVER in effect, the data read would be incorrect. The INFILE MISSOVER option instructs input to set to missing any values that are not found by the end of the record.

```
data readings;
 infile datalines missover;
 input Name $ R1-R6;
 datalines;
Gus  22 44 55 33
Gaia 24 22 23 31 76 31
;
proc print data=readings;
 title 'Readings';
 run;
```

Readings

Obs	Name	R1	R2	R3	R4	R5	R6
1	Gus	22	44	55	33	.	.
2	Gaia	24	22	23	31	76	31

You can also use the STOPOVER option in the INFILE statement. This causes the DATA step to halt execution when an INPUT statement does not find enough values in a record.

```
infile datalines stopover;
```

READING DELIMITED DATA

By default, the delimiter to read input data records with list input is a blank space. Both the delimiter-sensitive data (DSD) option and the DELIMITER= option affect how list input handles delimiters. The DELIMITER= option specifies that the INPUT statement use a character other than a blank as a delimiter for data values that are read with list input. When the DSD option is in effect, the INPUT statement uses a comma as the default delimiter. If the data contains two consecutive delimiters, the DSD option will treat it as two values whereas DELIMTER treats it as a single unit.

In the example below, the data values are separated by the double quote character and commas. DLM can specify those two delimiters and read the data correctly.

```
data address;
 infile datalines dlm='",';
 length city $10;
 input name $ age city $;
 datalines;
"Steve",32,"Monona"
"Tom",44,"Milwaukee"
"Kim",25,"Madison"
;
proc print data=address;
 title 'Address';
run;
```

```
                Address

Obs    city          name      age

 1     Monona        Steve      32
 2     Milwaukee     Tom        44
 3     Madison       Kim        25
```

In the next example, the two commas would be not be handled correctly with DLM alone. DSD treats consecutive as well as delimiters imbedded insite quotes correctly.

```
data address2;
 infile datalines dsd;
 length city $10;
 input name $ age city $;
 datalines;
"Steve",32,"Monona"
"Tom",44,"Milwaukee"
"Kim",,"Madison"
run;
proc print data=address2;
title 'Address2';
run;
```

```
Address2

Obs    city          name      age

 1     Monona        Steve      32
 2     Milwaukee     Tom        44
 3     Madison       Kim         .
```

SCANNING VARIABLE-LENGTH RECORDS FOR A SPECIFIC CHARACTER STRING

This example shows how to use TRUNCOVER in combination with SCANOVER to pull phone numbers from a phone book. The phone number is always preceded by the word "phone:". Because the phone numbers include international numbers, the maximum length is 32 characters.

```
filename phonebk host-specific-path;
data _null_;
  file phonebk;
  input line $80.;
  put line;
  datalines;
    Jenny's Phone Book
    Jim Johanson phone: 619-555-9340
       Jim wants a scarf for the holidays.
    Jane Jovalley phone: (213) 555-4820
       Jane started growing cabbage in her garden.
       Her dog's name is Juniper.
    J.R. Hauptman phone: (49)12 34-56 78-90
       J.R. is my brother.
   ;
run;
```

Use @'phone:' to scan the lines of the file for a phone number and position the file pointer where the phone number begins. Use TRUNCOVER in combination with SCANOVER to skip the lines that do not contain 'phone:' and write only the phone numbers to the log.

```
data _null_;
   infile phonebk truncover scanover;
   input @'phone:' phone $32.;
   put phone=;
run;
```

Partial SAS Log:

```
phone=619-555-9340
phone=(213) 555-4820
phone=(49)12 34-56 78-90
```

READING FILES THAT CONTAIN VARIABLE-LENGTH RECORDS

This example shows how to use LENGTH=, in combination with the $VARYING. informat, to read a file that contains variable-length records. When a record is read, the variable LINELEN contains the length of the current record. It can be then used to compute the length of the remaining variable and to read the rest of the record into secondvar.

```
data a;                                    /* SAS data step              */
   infile file-specification               /* input file                 */
          length=linelen;                  /* return length from input   */
   input firstvar 1-10 @;                  /* read first 10.             */
   varlen=linelen-10;                      /* Calculate VARLEN           */
   input @11 secondvar $varying500. varlen; /* read up to 500, using varlen*/
run;                                       /* end of step
```

LISTING THE POINTER LOCATION

INPUT keeps track of the pointer to the next byte in the buffer to be read. If there are multiple lines read on each pass, N= indicates how many lines are available to the pointer. The LINE= and COL= INFIILE options can pass the current value of the line and column pointers respectively, back to the data step.

```
data _null_;                          /* no dataset needed              */
 infile datalines n=2                 /* infile, two lines of buffer    */
        line=Linept col=Columnpt;     /* save line and column pointers  */
 input      name $ 1-15               /* read first line of group       */
      #2 @3 id;                       /* next line of group             */
 put linept= columnpt=;               /* display where input pointer is */
datalines;                            /* instream data                  */
J. Brooks
  40974
T. R. Ansen
  4032
;                                     /* end of data                    */
run;                                  /* end of step                    */
```

These statements produce the following log lines as the DATA step executes:

```
Linept=2 Columnpt=9
Linept=2 Columnpt=8
```

READING BINARY FILES

Binary files can be read one record at a time. Since there is no record terminator, we must specify the record length and indicate that each record is fixed length. This can be especially useful to read binary files on a Windows or UNIX system that originated on a Z/OS mainframe. The INFILE options along with appropriate Z/OS informats will read the record correctly.

For example: read a binary file of lrecl=33 that was built on a Z/OS machine.

```
data x;                               /* build a sas data set            */
 infile testing lrecl=33 recfm=f;     /* fixed record, 33 bytes at a time */
  input @1  id    s370fpd5.           /* packed decimal field            */
        @6  st    $ebcdic2.           /* s370 ebcdic                     */
        @8  ecode $ebcdic5.           /* s370 ebcdic                     */
        @13 qty1  s370fzd5.           /* s370 numeric                    */
        @18 qty2  s370fpd4.4          /* s370 pd                         */
        @22 desc  $ebcdic10.          /* s370 ebcdic                     */
        @32 node  s370fpd1.           /* s370 pd                         */
        @33 bunc  s370fpd1. ;         /* 3370 pd                         */
run;                                  /* end of step                     */
```

DYNAMICALLY SPECIFYING INPUT FILE NAMES

The INFILE FILEVAR= option specifies a variable that contains the complete name of the file to be read. Each time the variables value changes INFILE will point to the new file for subsequent INPUT. This technique allows a way to read many different files including all the members in a library or directory, or just to dynamically choose the files to read. Note that even though a file specification is required by the INFILE statement, it is not actually used as the file name is specified in the FILEVAR= variable. The FILENAME= variable will be set by SAS when the file changes and can be interrogated or displayed as desired.

This DATA step uses FILEVAR= to read from a different file during each iteration of the DATA step:

```
data allsales;
 length fileloc myinfile $ 300;      /* define vars for dsn       */
 input fileloc $ ;                   /* read instream data        */
 infile indummy filevar=fileloc      /* open file named in fileloc*/
                filename=myinfile    /* give filename reading     */
                end=done;            /* true when reading last rec*/
 do while(not done);                 /* do until no more recs     */
   input name $ jansale              /* read variables            */
         febsale marsale;            /* more                      */
   output;                           /* output to allsales        */
 end;                                /* next rec this file        */
 put 'Finished reading ' myinfile=;  /* display msg for fun       */
 datalines;
c:\temp\file1.dat
c:\temp\file2.dat
c:\temp\file3.dat
;                                    /* end of data               */
run;                                 /* end of data step          */
```

Partial SAS log:

```
NOTE: The infile INDUMMY is:
      File Name=c:\temp\file1.dat,
      RECFM=V,LRECL=256

Finished reading myinfile=c:\temp\file1.dat

NOTE: The infile INDUMMY is:
      File Name=c:\temp\file2.dat,
      RECFM=V,LRECL=256

Finished reading myinfile=c:\temp\file2.dat

NOTE: The infile INDUMMY is:
      File Name=c:\temp\file3.dat,
      RECFM=V,LRECL=256

Finished reading myinfile=c:\temp\file3.dat

NOTE: 1 record was read from the infile INDUMMY.
      The minimum record length was 11.
      The maximum record length was 11.
NOTE: 1 record was read from the infile INDUMMY.
      The minimum record length was 11.
      The maximum record length was 11.
NOTE: 1 record was read from the infile INDUMMY.
      The minimum record length was 11.
      The maximum record length was 11.
NOTE: The data set WORK.ALLSALES has 3 observations and 4 variables.
```

ACCESSING FILE NAMES WITH 'WILD CARDS'

INFILE file specification can be a "wild card" appropriate for your operating system. INFILE will point to all matching files and read all records from each of them. The EOV= option names a variable that SAS sets to 1 when the first record in a file in a series of concatenated files is read. The variable is set only after SAS encounters the next file, so it is never true for the first file. The END= options sets the corresponding variable to true on the last record of all. Again the FILENAME= variable can be used to determine the file being read. Note that if the directory being read

contains subdirectories, the job will fail with a message about not having enough authority to read. If this happens, another technique would be needed to read the files.

```
data allsales;
 length  myinfile $ 300;            /* define vars for dsn       */
 infile 'c:\temp\file*.dat'         /* open all matching files   */
        filename=myinfile           /* give filename reading     */
        eov=first_this_file         /* true 1st rec of each file */
                                    /* except first file         */
        end=done;                   /* true when reading last rec*/
 input name $ jansale               /* read variables            */
       febsale marsale;             /* more                      */
 savefile=myinfile;                 /* save in normal variable   */
 if _n_ =1 or                       /* first rec first file      */
   first_this_file then             /* or first rec all others   */
   put 'Start reading ' myinfile=;  /* display msg for fun       */
 run;


NOTE: The infile 'c:\temp\file*.dat' is:
      File Name=c:\temp\file1.dat,
      File List=c:\temp\file*.dat,RECFM=V,LRECL=256

NOTE: The infile 'c:\temp\file*.dat' is:
      File Name=c:\temp\file2.dat,
      File List=c:\temp\file*.dat,RECFM=V,LRECL=256

NOTE: The infile 'c:\temp\file*.dat' is:
      File Name=c:\temp\file3.dat,
      File List=c:\temp\file*.dat,RECFM=V,LRECL=256

Start reading myinfile=c:\temp\file1.dat
Start reading myinfile=c:\temp\file2.dat
Start reading myinfile=c:\temp\file3.dat

NOTE: 1 record was read from the infile 'c:\temp\file*.dat'.
      The minimum record length was 11.
      The maximum record length was 11.
NOTE: 1 record was read from the infile 'c:\temp\file*.dat'.
      The minimum record length was 11.
      The maximum record length was 11.
NOTE: 1 record was read from the infile 'c:\temp\file*.dat'.
      The minimum record length was 11.
      The maximum record length was 11.
NOTE: The data set WORK.ALLSALES has 3 observations and 5 variables.
```

SPECIFYING AN ENCODING WHEN READING AN EXTERNAL FILE

If files being read don't use the normal encoding schemes of ASCII or EBCDIC, a different encoding scheme can be specified. This example creates a SAS data set from an external file. The external file's encoding is in UTF-8, and the current SAS session encoding is Wlatin1. By default, SAS assumes that the external file is in the same encoding as the session encoding, which causes the character data to be written to the new SAS data set incorrectly.

```
libname myfiles 'SAS-data-library';
filename extfile 'external-file';
data myfiles.unicode;
   infile extfile encoding="utf-8";
   input Make $ Model $ Year;
run;
```

PLATFORM CONSIDERATIONS

Each of the platforms where SAS runs provides numerous INFILE options to process special features of the operating system. This is usually related to features of the file system but does include some other features as well.

One very useful application is reading files originally built on UNIX on a Windows platform and vice versa. The normal line end for Windows text files is a Carriage Return character and a Line Feed character. The Unix record terminator is only the Carriage Return character This is of course very confusing especially since the control characters are unprintable, but unless the terminator characters are known and coded for, the data may be read incorrectly.

In the example below, a Unix program is reading a file originally built under Windows. The TERMSTR=CRLF INFILE option tells the program to return records when both a carriage return and a line feed character are found. Without this option, the carriage return character would be read as the last character in the record.

```
data filefromwin;                          /* build a SAS ds            */
 infile '/some_unix.txt' termstr=crlf;     /* text file ended with CRLF */
 input Name $ Age Rate;                    /* input as normal           */
run;                                       /* run                       */
```

The opposite case is a Windows program reading a text file built on a Unix system. Since Unix only uses the linefeed character as a record terminator, we need to communicate this to INFILE.

```
data filefrounix;                          /* build a SAS ds            */
 infile '\some_win.txt' termstr=lf;        /* text file ended with LF   */
 input Name $ Age Rate;                    /* input as normal           */
run;                                       /* run                       */
```

ADDITIONAL WINDOWS AND UNIX OPTIONS

Additional INFILE options are available for encoding and other file handling. The various values for RECFM= allow for reading many types of files including Z/OS variable blocked files and much more. Please refer to the operating system documentation for INFILE for more details.

Z/OS OPTIONS

There are extensive INFILE options that apply to features found in Z/OS. Special options are also available for accessing ISAM, VSAM, IMS and other special Z/OS files. In addition there are options to access system control blocks.
The JFCB (job file control block) is a system block of 176 bytes allocated for every dd statement in a Z/OS job. The jfcb contains information such as datasetname, device type, catalog status, SYSIN or SYSOUT status, label processing options, and create date. The following are possible uses for the JFCB:

- accessing dataset name from JCL for titles
- determining whether the program is reading a live VSAM file, a sequential backup disk file, or a tape file,
- determining the file creation date.

Since this is a system control block, layout documentation is needed and the program may need to do bit testing.

The following example uses the JFCB to determine the DSNAME and DSORG.

```
data _null_;                        /* don't need dataset */
 infile in jfcb=jfcbin;             /* ask for jfcb       */
 length titldsn $ 44;               /* set lengths as     */
 length dsorg1 $1.;                 /* required           */
 if _n_ = 1 then                    /* first time in ?    */
  do;                               /* yes, do block      */
   titldsn=substr(jfcbin,1,44);     /* extract dsname     */
   dsorg1=substr(jfcbin,99,1);      /* and dsorg byte 1   */
    if dsorg1='.1......'b then      /* bit test as needed */
        dsorgout='PS';              /* must be sequential */
  end;                              /* end of block       */
 input etc. ;                       /* rest of program    */
  . . .
 retain titldsn dsorgout;           /* retain             */
run;                                /* end of step        */
```

Another system file called a VTOC (volume table of contents) is a file on each Z/OS disk pack containing information about the files on that disk. Data set characteristics and other system information can be gathered from the VTOC. Special VTOC options are available on the INFILE statement to read VTOCS.
The MAPDISK program shown below from the SAS sample library reads VTOCs.

```
/********* mapdisk ************************************************/
/*  this program reads the dscbs in a vtoc and produces a listing      */
/* of all data sets with their attributes and allocation data. the     */
/* volume to be mapped must be described by a disk dd stmt.:           */
/* //disk dd disp=shr,unit=sysda,vol=ser=xxxxxx                       */
/***********************************************************************/
data free(keep=loc cyl track total f5dscb)
     dsn (keep=dsname created expires lastref lastmod
          count extents dsorg recfm1 recfm4 aloc blksize
          lrecl secaloc tt r tracks volume)
     fmt1(keep=dsname created expires lastref lastmod
          count extents dsorg recfm1 recfm4 aloc blksize
          lrecl secaloc tt r tracks volume cchhr)
     fmt2(keep=cchhr tocchhr)
     fmt3(keep=cchhr alloc3);  length default=4;
  retain trkcyl 0;         /* error if no format 4 encountered */
  length volume volser1 $ 6  cchhr cchhr1 $ 5 ;
  format cchhr cchhr1 $hex10. dscbtype $hex2. ;

********read dscb and determine which format*******;
  infile disk vtoc cvaf cchhr=cchhr1 volume=volser1
         column=col ;
  input @45 dscbtype $char1. @;  volume=volser1;
  cchhr=cchhr1;
  if dscbtype='00'x then do; null+1;
        if null>200 then stop;
        return;  end;  null=0;
      if dscbtype= '1' then goto format1;
      if dscbtype= '2' then goto format2;
      if dscbtype= '3' then goto format3;
      if dscbtype= '5' then goto format5;
      if dscbtype= '4' then goto format4;
      if dscbtype= '6' then return;
      _error_=1;return;
format1:                    ****regular dscb type****;
  input @1 dsname $char44.
    . . .
```

THE POWER OF THE SAS FILENAME STATEMENT

Even though the INFILE and FILE statements have tremendous power, even more capabilities are available by utilizing the SAS FILENAME statement. The primary purpose of filename is to assign a nickname (fileref) to a single file. FILENAME can also assign the fileref to a directory or library in which case INFILE can read members by using parentheses in the dataset name. The SAS documentation refers to directories or partitioned data sets as "aggregate" storage areas. There are also default extensions assumed if the member name is not quoted. The following example reads a file which is a member of a Windows directory.

```
filename mytemp 'c:\temp';          /* assign a fileref to directory*/
data allsales;                      /* build a SAS ds               */
 infile mytemp("file1.dat");        /* open all matching files      */
 input name $ jansale               /* read variables               */
       febsale marsale;             /* more                         */
 run;                               /* end of step                  */
```

Assigning a filref to a directory can also allow data step functions to open the directory and interrogate it to find number of members, member names and more. The following program uses the DOPEN function to open the directory, the DNUM function to determine the number of members, DREAD to return the file name of each member, and the PATHNAME function to return the path of the directory. Those items along with the FILEVAR= and FILENAME= INFILE options can be used to read all the members in a directory in a different way than we saw earlier.

```
%let mfileref=temp;
filename &mfileref 'c:\temp';
/*****************************************************************************/
/* Loop thru all entries in directory and read each file found.              */
/* ************************************************************************* */
data x;
  did=dopen("&mfileref");                          /* try to open directory    */
  if did = 0 then                                  /*  if error opening file   */
     putlog "ERROR: Directory not found for mfilenames &mfileref";
  else                                             /* if directory opened ok.  */
     do;                                           /* do block                 */
        memberCount = dnum( did );                 /* get number of members    */
          do j = 1 to memberCount;                 /* for each member in       */
             fileName = dread( did, j );           /* directory.               */
             do;                                   /* do this                  */
                pathname=pathname("&mfileref");    /* for this program.        */
                flow=filename;                     /* grab flow, path_flow     */
                path_flow=trim(pathname)!!'\'!!filename;
                infile dummyf filevar=path_flow    /* point to file  file      */
                              filename=myinfile    /* give filename reading    */
                              end=eof lrecl=80 pad;/* mark end of file, lrecl  */
                do until(eof);                     /* loop thru all records    */
                   input name $ jansale            /* read variables           */
                         febsale marsale;          /* more                     */
                         savefile=myinfile;        /* save in normal variable  */
                         ;                         /* end of input             */
                   output;                         /* output to file           */
                end;                               /* end read all pgm lines   */
             end;                                  /* end if index >0          */
          end;                                     /* end loop thru all pgms   */
          did = close(did);                        /* close program directory  */
     end;                                          /* if directory opened      */
  stop;                                            /* stop data step.          */
run;                                               /* end of step              */
```

Partial SAS Log:

```
NOTE: The infile DUMMYF is:
      File Name=c:\temp\file1.dat,
      RECFM=V,LRECL=80

NOTE: The infile DUMMYF is:
```

```
      File Name=c:\temp\file2.dat,
      RECFM=V,LRECL=80

NOTE: The infile DUMMYF is:
      File Name=c:\temp\file3.dat,
      RECFM=V,LRECL=80

NOTE: 1 record was read from the infile DUMMYF.
      The minimum record length was 11.
      The maximum record length was 11.
NOTE: 1 record was read from the infile DUMMYF.
      The minimum record length was 11.
      The maximum record length was 11.
NOTE: 1 record was read from the infile DUMMYF.
      The minimum record length was 11.
      The maximum record length was 11.
NOTE: The data set WORK.X has 3 observations and 11 variables.
```

READING FROM AN FTP SITE

Not only can FILENAME point to files on your current machine, but using the FTP options our programs can read and write data to any authorized FTP server. No extra products besides base SAS and FTP are required. Programs can be very flexible by reading the data from other computers, but keep in mind that there will be transfer time to move the data from the other machine.
This example reads a file called sales in the directory /u/kudzu/mydata from the remote UNIX host hp720:

```
filename myfile ftp 'sales' cd='/u/kudzu/mydata'
         user='guest' host='hp720.hp.sas.com'
         recfm=v prompt;

data mydata / view=mydata;   /* Create a view */
   infile myfile;
   input x $10. y 4.;
run;i

proc print data=mydata;     /* Print the data */
run;
```

READING FROM A URL

FILENAME can also point to a web page that contains data that we might be interested in. This effectively opens our program to millions of online files.
This example reads the first 15 records from a URL file and writes them to the SAS log with a PUT statement:

```
filename mydata url
    'http://support.sas.com/techsup/service_intro.html';

data _null_;
   infile mydata length=len;
   input record $varying200. len;
   put record $varying200. len;
   if _n_=15 then stop;
run;
```

FILE STATEMENT OVERVIEW

The FILE statement in most respects is the opposite of the INFILE statement. Its function is to define an output raw file. Many of the options are the same for INFILE and FILE processing, except that the direction of data is coming out of the DATA step. Since the FILE statement can also define reports in SAS, however, there are some unique options for that purpose. The SAS FILE statement also can interact with ODS to route reports to different destinations. Since so many INFILE options have already been covered, those options that work the same way for FILE will not be covered again.

By default, PUT statement output is written to the SAS log. Use the FILE statement to route this output to either the same external file to which procedure output is written or to a different external file. You can indicate whether or not carriage control characters should be added to the file.

You can use the FILE statement in conditional (IF-THEN) processing because it is executable. You can also use multiple FILE statements to write to more than one external file in a single DATA step.

BASIC FILE SYNTAX

FILE *file-specification* <*options*> <*operating-environment-options*>;

file-specification

identifies an external file that the DATA step uses to write output from a PUT statement. File-specification can have these forms:

'external-file'

specifies the physical name of an external file which is enclosed in quotation marks. The physical name is the name by which the operating environment recognizes the file.

fileref

specifies the fileref of an external file. (note: The fileref must be previously associated with an external file in a FILENAME statement, FILENAME function, or a system command.)

fileref(file)

specifies a fileref of an aggregate storage location and the name of a file or member, enclosed in parentheses, that resides in that location.

LOG
is a reserved fileref that directs the output produced by any PUT statement to the SAS log.

PRINT
is a reserved fileref that directs the output produced by any PUT statement to the same file as the output that is produced by SAS procedures.

REPORT WRITING WITH FILE AND PUT

The DATA step can produce any report. Some of the DATA step programming features are

- FILE statement points to the report (or file) being written
- PUT statement does the actual writing
- pointers, formats, and column outputs are available
- end of file indicators, control break indicators
- has automatic header routines
- N=PS option allows access to full page at one time (default is one line or record at a time)
- All the DATA step programming power

WRITING TO THE SAS LOG

The default file is the SAS log file. The PUTLOG statement always writes to the log and the PUT statement writes to the most recently executed FILE statement. Since the SAS log contains SAS notes and other messages along with our output, the report may not be nice looking. It is, however, an excellent way to debug your DATA step logic by displaying variable information.

```
data _null_ ;
  infile rawin;
  input  @1  Name     $10.
         @12 Division $1.
         @15 Years     2.
         @19 Sales     7.2
         @28 Expense   7.2
         @36 State    $2.;
  put name division= expense=;
run;
```

The Resulting Log:

```
217  data _null_ ;
218    infile rawin;
219    input  @1  Name     $10.
220           @12 Division $1.
221           @15 Years     2.
222           @19 Sales     7.2
223           @28 Expense   7.2
224           @36 State    $2.;
225    put name division= expense=;
226  run;

CHRIS Division=H Expense=94.12
MARK Division=H Expense=52.65
SARAH Division=S Expense=65.17
PAT Division=H Expense=322.12
JOHN Division=H Expense=150.11
 . . .
```

THE SAS PRINT FILE

FILE PRINT directs PUT to write to the SAS print file. SAS will also use any TITLES, DATE, and NUM options at the top of each page. The FILE statement must precede the PUT statement.

```
title 'Softco Regional Sales';
data _null_;
 infile rawin;
 file print;
 input  @1  Name     $10.
       @12  Division  $1.
       @15 Years       2.
       @19 Sales       7.2
       @28 Expense     7.2
       @36 State      $2.;
 put 'Employee name '
      name   ' had sales of '
      sales;
 run;
```

The Resulting PRINT File

```
            Softco Regional Sales

   Employee name CHRIS  had sales of 233.11
   Employee name MARK  had sales of 298.12
   Employee name SARAH  had sales of 301.21
   Employee name PAT  had sales of 4009.21
   Employee name JOHN  had sales of 678.43
   . . .
```

FILE STATEMENT OPTIONS

Just as input files required more options and information as they became more complex, the same holds true for output files. A complete list of FILE options from the SAS documentation is listed below. You will notice that many of the options listed are the same as were available for INFILE. Again these options are available in all operating environments and operating system specific options are available. In addition, because the FILE statement may be producing a printed report, there are several options used exclusively for report writing.

ENHANCING THE REPORT

The FILE NOTITLES option turns off SAS TITLES and the HEADER= option names a LABEL to branch to whenever a new page is printed.

```
data _null_;
  infile rawin;
   input  @1  Name     $10.    @12 Division $1.
          @15 Years     2.     @19 Sales     7.2
          @28 Expense   7.2    @36 State    $2.;

   file print notitles header=headrtne;
   put @5 name $10.        @15 division $1. years 25-26
       @30 sales  8.2  @40 expense 8.2  @53 state $2.;
  return;
 headrtne:
   put @5 'Name' @12 'Division' @24 'Years' @33 'Sales'  @41 'Expense' @51 'State';
 return;
run;
```

The Resulting Output

```
   Name    Division    Years     Sales    Expense   State
   CHRIS      H          2      233.11     94.12     WI
   MARK       H          5      298.12     52.65     WI
   SARAH      S          6      301.21     65.17     MN
   PAT        H          4     4009.21    322.12     IL
   JOHN       H          7      678.43    150.11     WI
   . . .
```

COUNTING REMAINING LINES ON THE PAGE

The LINESLEFT= option will automatically count the remaining lines on the page so that your program can skip to a new page, etc.

```
data _null_;
 set softsale end=lastobs;
 by division;
```

```
 file print notitles linesleft=lineleft header=headrtne;
 if first.division then
    do;
       salesub=0;  expsub=0;
       if lineleft < 14 then put _page_;
    end;
 put @9 division $1.     @20 name $10.         @35 years 2.
     @45 sales comma8.2  @60 expense comma8.2;
    . . .
```

A FULL PAGE REPORT

N=PS gives access to an entire page before actually printing. Using variables for line and column pointers lets us fill several file buffers and print when completely filled in.

```
data _null_;
  retain Hlineno Slineno 4;
  set softsale;
  file print notitles header=headrtne n=ps;
  if division='H' then do;
    hlineno+1;
    put #hlineno  @1 name $10. +2 division +2 sales 7.2;
  end;
  if division='S' then do;
    slineno+1;
    put #slineno  @40 name $10. +2 division +2 sales 7.2;
  end;
return;
headrtne:
 put @7  'Division H' @46 'Division S' / ;
 put @1  'name       div    sales'
     @40 'name       div    sales' ;
 return;
  run;
```

```
  Division H                          Division S

  name       div    sales             name       div    sales

  BETH        H   4822.12             ANDREW      S   1762.11
  CHRIS       H    233.11             BENJAMIN    S    201.11
  JOHN        H    678.43             JANET       S     98.11
  MARK        H    298.12             JENNIFER    S    542.11
  PAT         H   4009.21             JOY         S   2442.22
  STEVE       H   6153.32             MARY        S   5691.78
  WILLIAM     H   3231.75             SARAH       S    301.21
                                      TOM         S   5669.12
```

WRITING RAW FILES

The ability to write raw files means that SAS can pass data to other programs. This can be used in any SAS DATA step that needs to pass on raw data. The rich DATA step programming language along with FILE options can serve as an extremely versatile utility program. Below are several examples of writing raw files with FILE and PUT.

MY FIRST SAS PROGRAMS

A file that only had good data in the first 20 of 80 columns needed to be packed together before sending the record down a communication line. That is, it needed to read 20 bytes of each group of four records and output one 80 byte record. On the other end of the line, a reversing program would reconstruct the original file layout. Using the FILE statement and a few other SAS statements made this an easy pair of programs to write.

```
DATA _NULL_;                        /* DON'T NEED DS     */
  INFILE IN;                        /* RAWFILE IN        */
  FILE OUT;                         /* RAWFILE OUT       */
  INPUT @1 TWENTY $CHAR20.;         /* READ 20 CHAR      */
  PUT      TWENTY $CHAR20. @;       /* 20 OUT, HOLD PTR  */
RUN;                                /* END OF STEP       */
```

THE REVERSING PROGRAM

PUT with trailing @@ holds the pointer across iterations and the new file is lrecl=80.

```
DATA _NULL_;                        /* DON'T NEED DATASET*/
 INFILE IN;                         /* RAW FILE IN       */
 FILE OUT LRECL=80;                 /* FILEOUT           */
 INPUT                              /* READ TWENTY/LOOP  */
     TWENTY  $CHAR20.               /* FIXED PTRS CAUSE  */
    @@ ;                            /* LOOPS, BE CAREFUL */
 IF TWENTY NE ' ' THEN              /* IF NONBLANK ?     */
       PUT                          /* OUTPUT 20         */
             @1 TWENTY  $CHAR20.; /* $CHAR SAVES BLANKS*/
 IF _N_ > 20 THEN                   /* A GOOD IDEA WHILE */
    STOP;                           /* TESTING           */
RUN;                                /* END OF STEP       */
```

A COPYING PROGRAM

Simply copy any sequential file.

```
DATA _NULL_;                       /* DON'T NEED DATASET */
  INFILE IN;                       /* RAW FILE IN        */
  FILE OUT;                        /* RAW FILE OUT       */
  INPUT;                           /* READ A RECORD      */
  PUT _INFILE_;                    /* WRITE IT OUT       */
RUN;                               /* END OF STEP        */
```

CHANGING DCB WHILE COPYING

Additional columns will be padded.

```
DATA _NULL_;                       /* DON'T NEED DATASET*/
  INFILE IN;                       /* RAW FILE IN       */
  FILE OUT LRECL=90                /* INCREASE DCB AS   */
           BLKSIZE=9000            /* NEEDED            */
           RECFM=FB;
  INPUT;                           /* READ A RECORD     */
  PUT _INFILE_;                    /* WRITE IT OUT      */
RUN;                               /* END OF STEP       */
```

A SUBSETTING PROGRAM

Select part of a file.

```
DATA _NULL_;                       /* DON'T NEED DATASET  */
  INFILE IN;                       /* RAW FILE IN         */
  FILE OUT;                        /* RAW FILE OUT        */
  INPUT @5 ID $CHAR1.;             /* INPUT FIELDS NEEDED */
  IF ID='2';                       /* WANT THIS RECORD?   */
  PUT _INFILE_;                    /* YEP, WRITE IT OUT   */
RUN;
```

SELECTING A RANDOM SUBSET

Randomly select about 10% of a file.

```
DATA _NULL_;                    /* NO DATASET NEEDED   */
  INFILE IN;                    /* RAW FILE IN         */
  FILE OUT;                     /* RAW FILE OUT        */
  INPUT;                        /* READ A RECORD       */
  IF RANUNI(0) LE .10;          /* TRUE FOR APP. 10%   */
  PUT _INFILE_;                 /* WRITE OUT OBS       */
RUN;                            /* END OF STEP         */
```

ADDING SEQUENCE NUMBERS

Write out a buffer, then overlay.

```
DATA _NULL_;                    /* NO DATASET NEEDED   */
  INFILE IN;                    /* RAW FILE IN         */
  FILE OUT;                     /* RAW FILE OUT        */
  INPUT;                        /* READ A RECORD       */
  SEQ=_N_*100;                  /* COMPUTE SEQ NO      */
  PUT _INFILE_                  /* OUTPUT INPUT REC    */
      @73 SEQ Z8.;              /* OVERLAY WITH SEQ    */
RUN;                            /* END OF STEP         */
```

WRITING LITERALS IN EVERY RECORD

Put 'SSC' in columns 10-12 of every line.

```
DATA _NULL_;                    /* NO DATASET NEEDED   */
  INFILE IN;                    /* RAW FILE IN         */
  FILE OUT;                     /* RAW FILE OUT        */
  INPUT;                        /* READ A RECORD       */
  PUT _INFILE_                  /* OUTPUT INPUT REC    */
      @10 'SSC';                /* OVERLAY WITH CONST. */
RUN;                            /* END OF STEP         */
```

PRINTING A FILE WITH CARRIAGE CONTROL

Report files, microfiche files etc. can be handled.

```
DATA _NULL_;                    /* DON'T NEED DATASET  */
  INFILE IN;                    /* INPUT FILE IN       */
  FILE PRINT NOPRINT;           /* DON'T ADD CC        */
  INPUT;                        /* READ A RECORD       */
  PUT _INFILE_;                 /* WRITE IT OUT        */
RUN;                            /* END OF STEP         */
```

CORRECTING A FIELD ON A FILE

Logic can be altered to match any situation.

```
DATA _NULL_;                    /* DON'T NEED DATASET  */
  INFILE IN;                    /* INPUT FILE IN       */
  FILE OUT;                     /* OUTPUT FILE         */
  INPUT @5 ID $CHAR1.;          /* INPUT FIELDS NEEDED */
  IF ID='2'                     /* CHANGE AS NEEDED    */
     THEN ID='3';
  PUT   _INFILE_                /* OUTPUT FILE         */
     @5 ID CHAR1.;              /* OVERLAY CHANGED ID  */
RUN;                            /* END OF STEP         */
```

FILE STATEMENT NOTES

As stated earlier many of options available on INFILE are also available with FILE. These include accessing the buffers, altering recfm, termstr, changing the output file dynamically, file encoding, and binary file writing. Using the FILENAME statement lets us write to email, FTP sites, aggregate file locations, and much more. Again, since I

covered those topics with INFILE above, I wont repeat them, but their usage can be just as valuable with the FILE statement for files that we need to create.

CONCLUSION

The INFILE and FILE statements along with the rest of SAS allow us to read and write virtually any type of file with minimal effort. This allows the other SAS components to access and provide data to any outside source or destination with minimal effort.

REFERENCES

The Syntax and many of the examples used were from the online SAS Documentation and help screens provided with SAS release 9.1

ACKNOWLEDGMENTS

Dr. Joy First for advice and support in the preparation of this document.

RECOMMENDED READING

Recommended reading lists go after your acknowledgments. This section is not required.

CONTACT INFORMATION

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

Name	Steven First, President
Enterprise	Systems Seminar Consultants
Address	2997 Yarmouth Greenway Drive
City, State ZIP	Madison, WI 53711
Work Phone:	608 278-9964
Fax:	608 278-0065
E-mail:	sfirst@sys-seminar.com
Web:	www.sys-seminar.com

APPENDIX A: INFILE SYNTAX

INFILE *file-specification <options > <operating-environment-options>* ;

INFILE *DBMS-specifications;*

file-specification

identifies the source of the input data records, which is an external file or instream data. *File-specification* can have these forms:

'*external-file*'

specifies the physical name of an external file. The physical name is the name that the operating environment uses to access the file.

fileref

specifies the fileref of an external file. (note: The fileref must be previously associated with an external file in a FILENAME statement, FILENAME function, or a system command)

fileref(file)

specifies a fileref of an aggregate storage location and the name of a file or member, enclosed in parentheses, that resides in that location.

Operating Environment Information: Different operating environments call an aggregate grouping of files by different names, such as a directory, a MACLIB, or a partitioned data set. Details are given in SAS operating system documentation.

CARDS | CARDS4
DATALINES | DATALINES4

DBMS-Specifications

enable you to read records from some DBMS files. You must license SAS/ACCESS software to be able to read from DBMS files. See the SAS/ACCESS documentation for the DBMS that you use.

INFILE OPTIONS

BLKSIZE=*block-size*

specifies the block size of the input file.

COLUMN=*variable*

names a variable that SAS uses to assign the current column location of the input pointer. Like automatic variables, the COLUMN= variable is not written to the data set.

EOV=*variable*

names a variable that SAS sets to 1 when the first record of a file in a series of concatenated files is read. The variable is set only after SAS encounters the next file. Like automatic variables, the EOV= variable is not written to the data set.

DELIMITER= *delimiter(s)*

specifies an alternate delimiter (other than a blank) to be used for LIST input, where *delimiter* is

'*list-of-delimiting-characters*'

specifies one or more characters to read as delimiters.

character-variable

specifies a character variable whose value becomes the delimiter.

ENCODING= '*encoding-value*'

specifies the encoding to use when reading from the external file. The value for ENCODING= indicates that the external file has a different encoding from the current session encoding.

When you read data from an external file, SAS transcodes the data from the specified encoding to the session encoding.

END=*variable*

names a variable that SAS sets to 1 when the current input data record is the last in the input file. Until SAS processes the last data record, the END= variable is set to 0. Like automatic variables, this variable is not written to the data set.

EOF=*label*

specifies a statement label that is the object of an implicit GO TO when the INFILE statement reaches end of file. When an INPUT statement attempts to read from a file that has no more records, SAS moves execution to the statement label indicated.

EXPANDTABS | NOEXPANDTABS

specifies whether to expand tab characters to the standard tab setting, which is set at 8-column intervals that start at column 9.

FILENAME=*variable*

names a variable that SAS sets to the physical name of the currently opened input file. Like automatic variables, the FILENAME= variable is not written to the data set.

FILEVAR=*variable*

names a variable whose change in value causes the INFILE statement to close the current input file and open a new one. When the next INPUT statement executes, it reads from the new file that the FILEVAR= variable specifies. Like automatic variables, this variable is not written to the data set. Restriction:

The FILEVAR= variable must contain a character string that is a physical filename. When you use the FILEVAR= option, the *file-specification* is just a placeholder, not an actual filename or a fileref that has been previously assigned to a file. SAS uses this placeholder for reporting processing information to the SAS log. It must conform to the same rules as a fileref. Use FILEVAR= to dynamically change the currently opened input file to a new physical file.

FIRSTOBS=*record-number*

specifies a record number that SAS uses to begin reading input data records in the input file.

FLOWOVER

causes an INPUT statement to continue to read the next input data record if it does not find values in the current input line for all the variables in the statement. This is the default behavior of the INPUT statement.

LENGTH=*variable*

names a variable that SAS sets to the length of the current input line. SAS does not assign the variable a value until an INPUT statement executes. Like automatic variables, the LENGTH= variable is not written to the data set.

LINE=*variable*

names a variable that SAS sets to the line location of the input pointer in the input buffer. Like automatic variables, the LINE= variable is not written to the data set. The value of the LINE= variable is the current relative line number within the group of lines that is specified by the N= option or by the #*n* line pointer control in the INPUT statement

LINESIZE=*line-size*

specifies the record length that is available to the INPUT statement.

If an INPUT statement attempts to read past the column that is specified by the LINESIZE= option, then the action taken depends on whether the FLOWOVER, MISSOVER, SCANOVER, STOPOVER, or TRUNCOVER option is in effect. FLOWOVER is the default.

Use LINESIZE= to limit the record length when you do not want to read the entire record.

If your data lines contain a sequence number in columns 73 through 80, then use this INFILE statement to restrict the INPUT statement to the first 72 columns:

```
infile file-specification linesize=72;
```

LRECL=*logical-record-length*

specifies the logical record length.

LRECL= specifies the physical line length of the file. LINESIZE= tells the INPUT statement how much of the line to read.

MISSOVER

prevents an INPUT statement from reading a new input data record if it does not find values in the current input line for all the variables in the statement. When an INPUT statement reaches the end of the current input data record, variables without any values assigned are set to missing.

Use MISSOVER if the last field(s) might be missing and you want SAS to assign missing values to the corresponding variable.

N=*available-lines*

specifies the number of lines that are available to the input pointer at one time.

The highest value following a # pointer control in any INPUT statement in the DATA step. If you omit a # pointer control, then the default value is 1.

This option affects only the number of lines that the pointer can access at a time; it has no effect on the number of lines an INPUT statement reads.

When you use # pointer controls in an INPUT statement that are less than the value of N=, you might get unexpected results. To prevent this, include a # pointer control that equals the value of the N= option. Here is an example:

```
infile 'externalfile' n=5;
input #2 name : $25. #3 job : $25. #5;
```

The INPUT statement includes a #5 pointer control, even though no data is read from that record.

NBYTE=*variable*

specifies the name of a variable that contains the number of bytes to read from a file when you are reading data in stream record format (RECFM=S in the FILENAME statement).

OBS=*record-number* | MAX

PAD | NOPAD

controls whether SAS pads the records that are read from an external file with blanks to the length that is specified in the LRECL= option.

PRINT | NOPRINT

specifies whether the input file contains carriage-control characters.

To read a file in a DATA step without having to remove the carriage-control characters, specify PRINT. To read the carriage-control characters as data values, specify NOPRINT.

RECFM=*record-format*

specifies the record format of the input file.

SCANOVER

causes the INPUT statement to scan the input data records until the character string specified in the @'*character-string*' expression is found.

SHAREBUFFERS

specifies that the FILE statement and the INFILE statement share the same buffer.

Use SHAREBUFFERS with the INFILE, FILE, and PUT statements to update an external file in place. This saves CPU time because the PUT statement output is written straight from the input buffer instead of the output buffer.

START=*variable*

names a variable whose value SAS uses as the first column number of the record that the PUT _INFILE_ statement writes. Like automatic variables, the START variable is not written to the data set.

STOPOVER

causes the DATA step to stop processing if an INPUT statement reaches the end of the current record without finding values for all variables in the statement.

TRUNCOVER

overrides the default behavior of the INPUT statement when an input data record is shorter than the INPUT statement expects. Use TRUNCOVER to assign the contents of the input buffer to a variable when the field is shorter than expected.

UNBUFFERED

tells SAS not to perform a buffered ("look ahead") read.

INFILE=*variable*

names a character variable that references the contents of the current input buffer for this INFILE statement. You can use the variable in the same way as any other variable, even as the target of an assignment. The variable is automatically retained and initialized to blanks. Like automatic variables, the _INFILE_= variable is not written to the data set.

Note: There are additional options for Z/OS, UNIX, and Windows platforms. See the SAS online documentation for a complete set of options.

APPENDIX B: FILE SYNTAX

FILE *file-specification* <*options*> <*operating-environment-options*>;

file-specification

identifies an external file that the DATA step uses to write output from a PUT statement. File-specification can have these forms:

'external-file'

specifies the physical name of an external file, which is enclosed in quotation marks. The physical name is the name by which the operating environment recognizes the file.

fileref

specifies the fileref of an external file. (note: The fileref must be previously associated with an external file in a FILENAME statement, FILENAME function, or a system command)

fileref(file)

specifies a fileref of an aggregate storage location and the name of a file or member, enclosed in parentheses, that resides in that location

LOG
is a reserved fileref that directs the output that is produced by any PUT statements to the SAS log.

PRINT
is a reserved fileref that directs the output that is produced by any PUT statements to the same file as the output that is produced by SAS procedures.

FILE STATEMENT OPTIONS

BLKSIZE=*block-size*
specifies the block size of the output file.

COLUMN=*variable*
specifies a variable that SAS automatically sets to the current column location of the pointer.

DELIMITER= 'string-in-quotation-marks' | character-variable
specifies an alternate delimiter (other than blank) to be used for LIST output. This option is ignored for other types of output (formatted, column, named).

DROPOVER
discards data items that exceed the output line length (as specified by the LINESIZE= or LRECL= options in the FILE statement).

DSD (delimiter sensitive data)

specifies that data values containing embedded delimiters, such as tabs or commas, be enclosed in quotation marks. The DSD option enables you to write data values that contain embedded delimiters to LIST output.

ENCODING= *'encoding-value'*
specifies the encoding to use when writing to the output file.

FILENAME=*variable*
defines a character variable, whose name you supply, that SAS sets to the value of the physical name of the file currently open for PUT statement output. The physical name is the name by which the operating environment recognizes the file.

FILEVAR=*variable*
defines a variable whose change in value causes the FILE statement to close the current output file and open a new one the next time the FILE statement executes. The next PUT statement that executes writes to the new file that is specified as the value of the FILEVAR= variable.

FLOWOVER
causes data that exceeds the current line length to be written on a new line.

FOOTNOTES | NOFOOTNOTES
controls whether currently defined footnotes are printed.

HEADER=*label*
defines a statement label that identifies a group of SAS statements that you want to execute each time SAS begins a new output page.

LINE=*variable*
defines a variable whose value is the current relative line number within the group of lines available to the output pointer.

LINESIZE=*line-size*
sets the maximum number of columns per line for reports and the maximum record length for data files.

LINESLEFT=*variable*
defines a variable whose value is the number of lines left on the current page.

LRECL=*logical-record-length*
specifies the logical record length of the output file.

MOD
writes the output lines after any existing lines in the file.

N=*available-lines*
specifies the number of lines that you want available to the output pointer in the current iteration of the DATA step.

ODS < = (*ODS-suboptions*) >
specifies to use the Output Delivery System to format the output from a DATA step.

OLD
replaces the previous contents of the file.

PAD | NOPAD
controls whether records written to an external file are padded with blanks to the length that is specified in the LRECL= option.

PAGESIZE=*value*
sets the number of lines per page for your reports.

PRINT | NOPRINT
controls whether carriage control characters are placed in the output lines.

RECFM=*record-format*
specifies the record format of the output file.

STOPOVER
stops processing the DATA step immediately if a PUT statement attempts to write a data item that exceeds the current line length.

TITLES | NOTITLES
controls the printing of the current title lines on the pages of files. When NOTITLES is omitted, or when TITLES is specified, SAS prints any titles that are currently defined.

FILE=*variable*
names a character variable that references the current output buffer of this FILE statement.