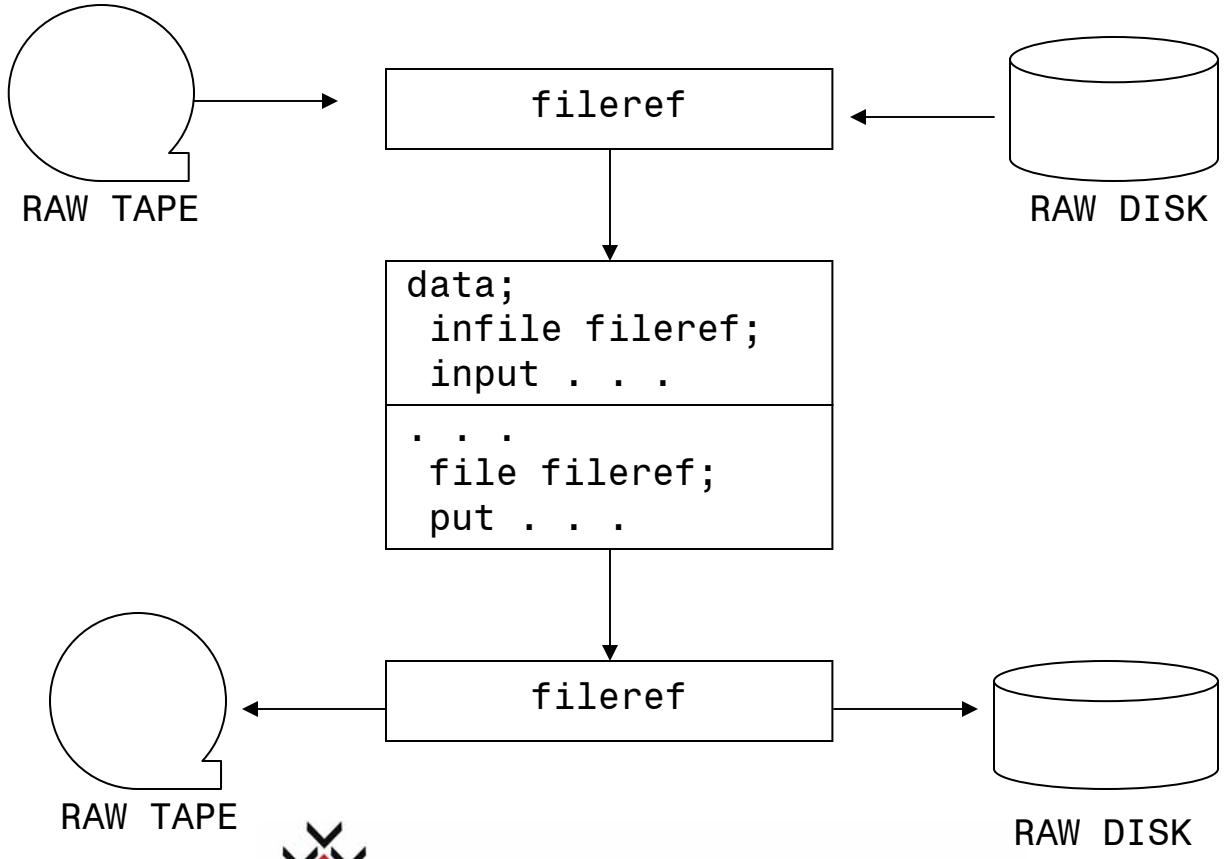
The SAS® INFILE and FILE Statements



SYSTEMS SEMINAR CONSULTANTS, INC.

Steven J. First, President
2997 Yarmouth Greenway Drive, Madison, WI 53711
Phone: (608) 278-9964, Web: www.sys-seminar.com

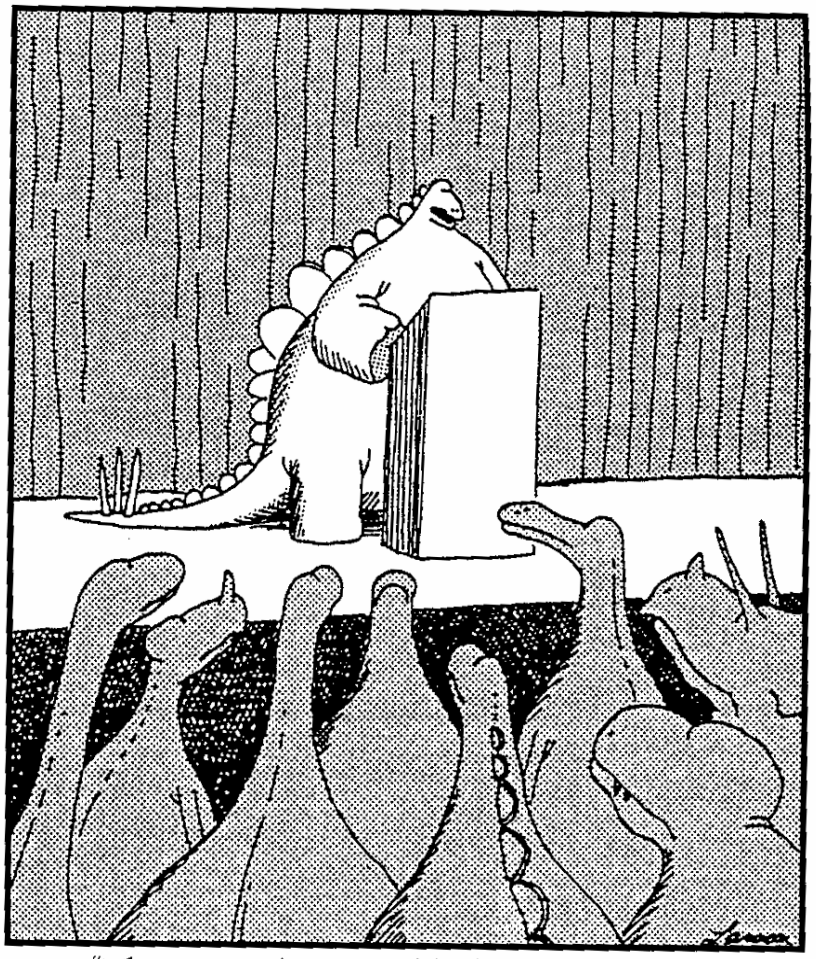
The SAS® INFILE and FILE Statements

This presentation was written by Systems Seminar Consultants, Inc.

SSC specializes SAS software and offers SAS:

- Training Services
- Consulting Services
- Help Desk Plans
- Newsletter subscriptions to *The Missing Semicolon™.*

Global Warming Part 1



"The picture's pretty bleak, gentlemen. …
The world's climates are changing, the mammals
are taking over, and we all have a brain
about the size of a walnut."

Abstract

SAS® INFILE and FILE Statements:

- allow us to link to raw files
- provide some of the most flexible features of the SAS system
- INPUT and PUT can read and write data
- provide many options, making reading and writing simple to complex files easy

Introduction

In any programming language, a link is needed between programs, files.

INFILE and FILE are statements that SAS uses:

- linking to *raw* files (normally contain only data and no data dictionary)
- INFILE is used to point t*o input* files, FILE points to *output* files

Other than the direction of data flow, INFILE and FILE act the same :

- many of the same options
- some unique options for INFILE versus FILE

Raw Data Sometimes Has Little Metatdata

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 information to read/write data
- allow SAS to process a rich and varied range of files with minimal effort

INFILE/FILE work with other SAS statements to provide extensive data input and output in the DATA step, such as:

- FILENAME
- DATALINES
- PUT
- INPUT

Basic INFILE Syntax

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

INFILE *DBMS-specifications;*

file-specification*:* identifies source of input data records (either external file or instream data)

Forms:

'*external-file*'

- specifies physical name of an external file, so system can access file

fileref

- specifies “nickname” of an external file
- must be previously associated with an external file (through FILENAME statement, FILENAME function, or system command)

fileref(file)

- specifies “nickname” for aggregate storage location
- name of a file or member residing in that location (enclosed in parentheses)

Basic INFILE Syntax (continued)

Operating Environment Information:

- different environments call an aggregate grouping of files by different names, such as a directory, a MACLIB, or a partitioned data set
- details 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

Example:

```
fileref in 'c:\temp\mydat.dat';     /* assign nickname       */
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           */
```

Note:

The same holds true for the FILE statement which must precede any PUT statement that performs the writing to the output raw file.

Reading Multiple Files

Read multiple files in a single data step:

- specify more than one INFILE statement
- step will stop when any file tries to read past end
- use INFILE statement (such as an IF-THEN statement) in conditional processing because it is executable
- enables control of input source

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.

Reading Multiple Files (continued)

Reads from two input files with each iteration:

- file remains open as SAS switches from one file to the next
- input pointer remains in place to read values with next INPUT

Example:

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

- DATA step terminates when SAS reaches end of file on shortest input file.

A HEX Listing Program

INPUT statement

- no need to specify variables to read
- reads line into buffer from input file
- can be useful with LIST statement to display raw input file's buffer
- LIST displays hexadecimal if unprintable characters present

Example:

```
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

Access or alter buffer area

- after an INFILE and INPUT statement executes
- use special variable called _INFILE_
- allows simple way to read without extensive coding

Example:

```
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             */
```

Altering the INPUT Buffer

We can actually alter values before reading individual fields.

Example:
The following file contains angle brackets 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>
```

Altering the INPUT Buffer (continued)

The following code:

- reads and holds record in input buffer using INPUT statement
- removes brackets (< >) from _INFILE_ field with compress function
- parses value in buffer using second INPUT statement
- displays the SAS variables with a PUT
- skips first header record because of FIRSTOBS INFILE option

```
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;
```

Altering the INPUT Buffer (continued)

The results are shown below.

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 Current Buffer

The _INFILE_=*variable* names a character variable:

- reference contents of current input buffer for INFILE statement
- variable not written to SAS dataset
- define this type of variable when inputting multiple files

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

- Process instream data or use other options with INFILE statement
- Read data with INPUT statement following DATALINES statement
- Option CARDIMAGE assumes 80 byte record padded with blanks
- OPTIONS NOCARDIMAGE may need to be specified for longer dataline input

Reading Instream Data Records with INFILE (cont)

```
data _null_;
   length city number $16. minutes charge 8;
   infile datalines firstobs=2 _infile_=phonebuff;
   input @;
...
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 current input record, the input pointer moves to column 1 of the next record

- governed by FLOWOVER option with message written to log
- useful in reading data that flows over into several lines of input.

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

```
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;
```

Reading Past the End of a Line (continued)

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

Who the heck are Gus and Gaia anyway???

Although, their fame may not reach San Antonio, they're celebrities where I come from.

Gus and Gaia, cautious buddies

Format Inputs to Control Short Records

With data that doesn’t flow over, the FLOWOVER behavior can cause errors.

Use these options to change INPUT statement behavior:

STOPOVER

- treats condition as error
- stops building data set

MISSOVER

- sets remaining INPUT statement variables to missing values

SCANOVER

- used with @'character-string'
- scans the input record until it finds specified *character-string*

TRUNCOVER

- variables are set to missing when records are short

FLOWOVER

- restores default behavior

Format Inputs to Control Short Records (cont.)

The TRUNCOVER and MISSOVER options are similar.

Both set remaining INPUT variables to missing

MISSOVER

- causes INPUT statement to set value to missing if unable to read an entire field because the field length that is specified is too short

TRUNCOVER

- writes characters read to last variable, remaining variables are missing

Format Inputs to Control Short Records (cont.)

Example (using input of variable-length):

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

Read the data to create a SAS data set, creating three observations from input records because by default the FLOWOVER option is used.

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

Obs	testnum
1	22
2	4444
3	55555

This output of course is not correct.

Format Inputs to Control Short Records (cont.)

INFILE MISSOVER sets not read values to missing.

- “set to missing” if over the end of record.
- All the values were read from short records set to missing.

```
infile 'external-file' missover;
```

INFILE TRUNCOVER

- tells INPUT to read as much as possible
- value will be chopped off.

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

Format Inputs to Control Short Records (cont.)

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:

- record width
- if line is shorter, pad

Program below reads data correctly as all recs 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

List Input: For Short Records and Missing Values

INFILE MISSOVER option instructs input to set to missing any values not found by the end of the record, rather than FLOWOVER.

The example below shows a data record that contain missing values.

```
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;
```

List Input: For Short Records and Missing Values

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

Use STOPOVER option with the INFILE statement.

- causes DATA step to halt execution
- when INPUT statement does not find enough values

Example:

```
infile datalines stopover;
```

Reading Delimited Data

Default: blank space is delimiter to read input data records with list input

Options to affect handling of delimeters by list input:

- **delimiter-sensitive data (DSD)**: INPUT uses comma as default delimiter
- **DELIMITER=** specifies to use other than blank as delimiter

Note:

If the data contains two consecutive delimiters, the DSD option will treat it as two values whereas DELIMITER treats it as a single unit.

Reading Delimited Data (continued)

For data values separated by two delimiters, use DLM to specify the characters used as delimiters and read the data correctly.

Example:

```
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;
```

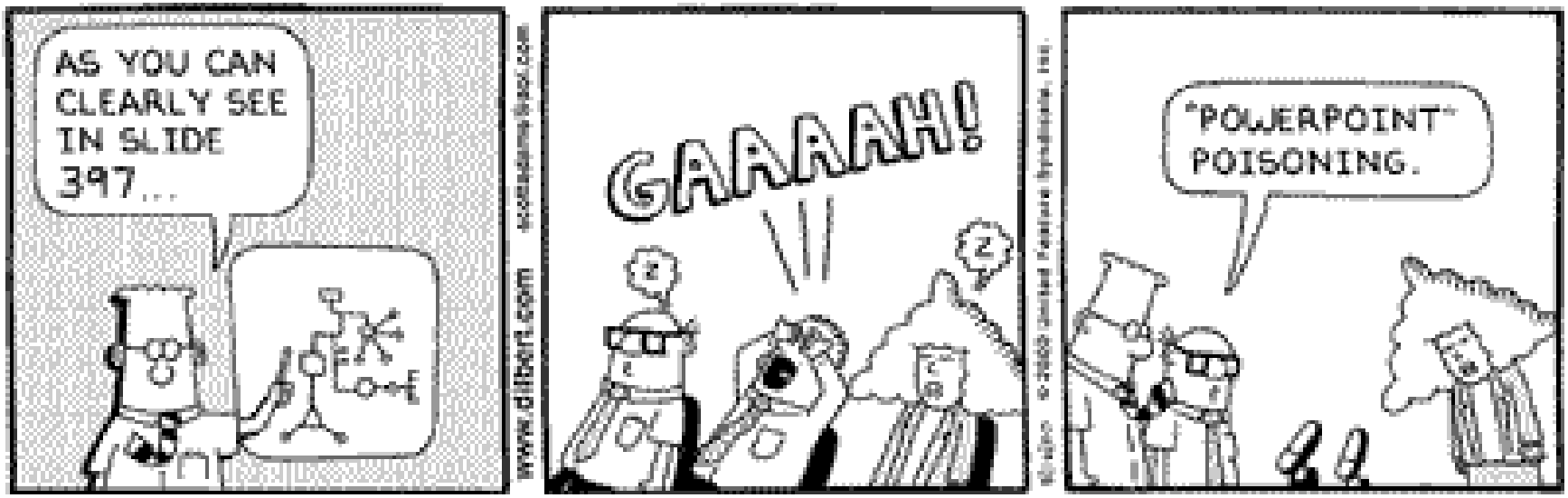
Reading Delimited Data (continued)

Results:

Address

Obs	city	name	age
1	Monona	Steve	32
2	Milwaukee	Tom	44
3	Madison	Kim	25

Losing the Crowd



Reading Delimited Data (continued)

Two commas would be not be handled correctly with DLM alone.
DSD treats consecutive and embedded delimiters correctly.

Example:

```
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;
```

Reading Delimited Data (continued)

Results

Address2

Obs	city	name	age
1	Monona	Steve	32
2	Milwaukee	Tom	44
3	Madison	Kim	.

Notes:

- DSD can be used in FILE statement to insert delimiters in output files.

Scan Variable-Length Records for Character String

A file that needs TRUNCOVER and SCANOVER to read phone numbers:

- number always preceded by word "phone:“
- maximum length is 32 character, because of inclusion of intl numbers

```
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.
```

Scan Variable-Length Records for Character String

Use @'phone:' to…

- scan file's lines for phone number
- position file pointer where phone number begins

Use **TRUNCOVER** in combination with **SCANOVER** to:

- skip lines that do not contain 'phone:'
- write only phone numbers to log

```
data _null_;
   infile 'path' 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
```

Read Files that Contain Variable Length Records

Read a file containing variable records with LENGTH= and $VARYING.

- when record read, LINELEN contains the length of the current record
- use LINELEN to compute length of remaining variables & read rest

Example:

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

Listing the Pointer Location

INPUT keeps track of the pointer to the next byte in buffer to be read.

N= sets buffer lines available to pointer

LINE=, COL= returns current line and column position of pointer

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

Listing the Pointer Location (continued)

Above program produces 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.

We must:

- not have a record terminator
- specify record length
- indicate record format is fixed length

This can be especially useful to read binary files that originated on a Z/OS mainframe on a Windows or UNIX system.

The INFILE options along with appropriate Z/OS informats will read the record correctly.

Reading Binary Files (continued)

Read a binary file of lrecl=33 that was built on a Z/OS machine.

Example:

```
data x;                              /* build sas data set  */
   infile testing                    /* fixed record,       */
      lrecl=33 recfm=f;              /* 33 bytes at a time  */
  input @1  id    s370fpd5.          /* packed decimal fld  */
        @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         */
```

Specify Input File Names Dynamically

infile *filespec* filevar=*variable* filename=*variable*;

- specifies variable (with complete name of file) to be read
- points to new file for subsequent INPUT when variables value changes
- reads many different files (including all members in library or directory)
- dynamically chooses files to read

File specification is

- required by INFILE statement
- not used as file name

FILENAME= variable

- set by SAS when file changes
- can be interrogated or displayed as desired

Specify Input File Names Dynamically (continued)

Use FILEVAR= to read from a different file during each iteration of the step:

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

Specify Input File Names Dynamically (continued)

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
```

Specify Input File Names Dynamically (continued)

Partial SAS log (continued):

```
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.
```

Access File Names with ‘Wild Cards’

INFILE file specification can be an appropriate “wild card”.

- points to all matching files
- read all records from each file

EOV= option

- names variable SAS sets to 1 when first record in file in a series of concatenated files is read
- variable set after SAS encounters next file,
- never true for first file

END= options

- sets corresponding variable to true on last record of all

FILENAME= variable

- used to determine file being read

If Directory contains Subdirectories…

- job will fail with message about not having authority to read
- another technique needed to read files

Access File Names with ‘Wild Cards’ (continued)

Example:

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

Access File Names with 'Wild Cards' (continued)

Partial SAS log:

```
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
```

Access File Names with ‘Wild Cards’ (continued)

Partial SAS log (continued):

```
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.
```

Specify an Encoding when Reading External Files

If files being read don’t use normal encoding schemes of ASCII or EBCDIC, a different encoding scheme can be specified.

Example:

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

Notes:

- creates SAS data set from external file
- file's encoding is in UTF-8 and current SAS session encoding is Wlatin1
- SAS assumes external file is in same encoding as session encoding, causing character data to be written incorrectly to new dataset.

Platform Considerations

Every SAS platform provides numerous INFILE options to process special features of the operating system.

INFILE options usually related to features of the file system, it does include some other features, such as reading files built on UNIX or a Windows platform and vice versa….

- normal line end for Windows text files is a Carriage Return character and a Line Feed character.
- Unix record terminator is only the Carriage Return character
- confusing since control characters are unprintable and data may be read incorrectly

Platform Considerations (continued)

A Unix program is reading a file originally built under Windows.

TERMSTR=CRLF INFILE option:

- tells program to return records when both a carriage return and a line feed character are found
- without option, reads carriage return character as last character in record

Example:

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

Platform Considerations (continued)

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, communicate this to INFILE.

Example:

```
data filefrounix;                          /* build a SAS ds  */
 infile ‘\some_win.txt’ termstr=lf;        /* LF ends txt     */
 input Name $ Age Rate;                    /* input as normal */
run;                                       /* run             */
```

Additional Windows and UNIX Options

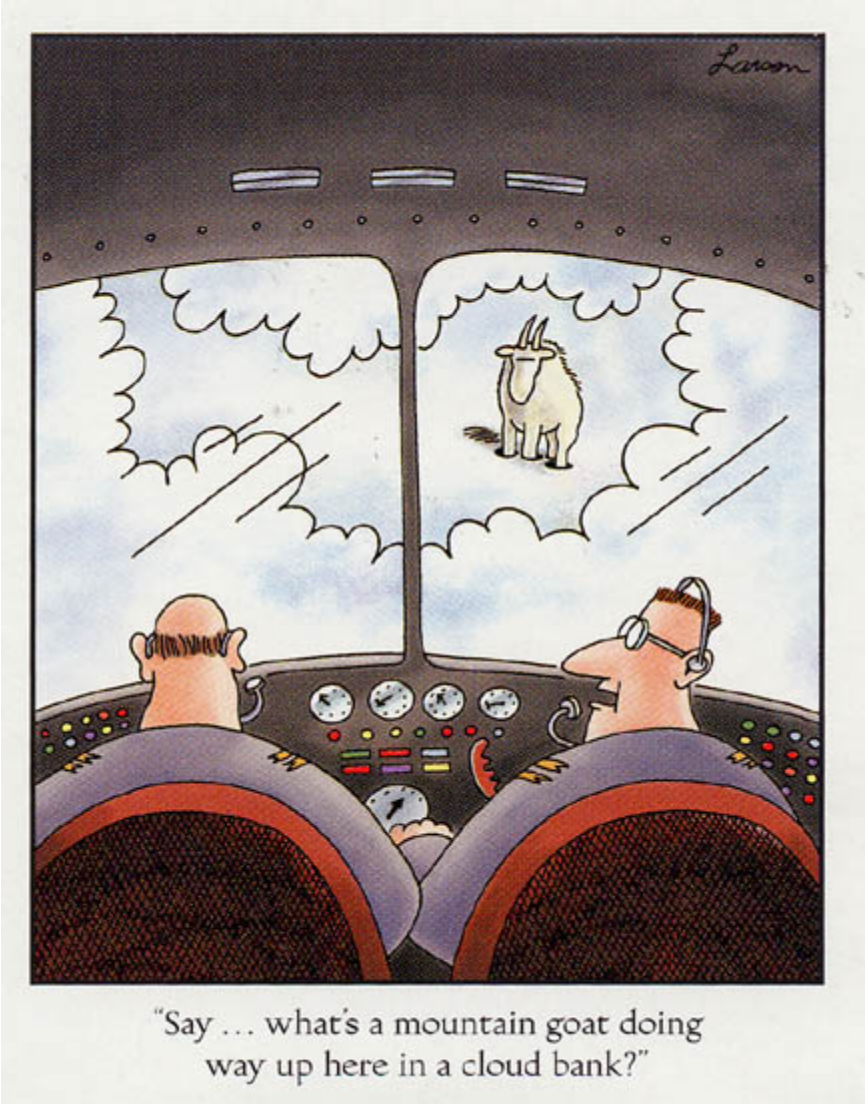
Additional INFILE options are available for encoding and other file handling.

- various values for RECFM= allow reading many types of files, including Z/OS variable blocked files and much more.

Note:

Please refer to the operating system documentation for more details on INFILE.

What You Don't Want On the Flight Home



Z/OS Options

There are extensive INFILE options that apply to features found in Z/OS.

- options available for accessing ISAM, VSAM, IMS and other special Z/OS files

JFCB (job file control block):

- option to access system control blocks
- system block of 176 bytes allocated for every dd statement in a Z/OS job.
- contains information such as dataset name, device type, catalog status, SYSIN or SYSOUT status, label processing options, and creation date

Possible Uses for the JFCB:

- access dataset name from JCL for titles
- determine if program reading live VSAM file, sequential backup disk file, or tape file
- determine file creation date

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

Z/OS Options (continued)

The following example uses the JFCB to determine 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         */
```

Z/OS Options (continued)

VTOC (volume table of contents)

- option to access system control blocks
- file on each Z/OS disk pack containing information about files on disk
- data set characteristics and other system information can be gathered from VTOC.
- special VTOC options are available on INFILE statement to read VTOCS

Z/OS Options (continued)

This MAPDISK program from the SAS sample library reads VTOCs:

```
/********* mapdisk ************************************/
/* this pgm reads dscbs in vtoc and produces listing   */
/* of datasets with attributes and allocation data. The*/
/* vol to be mapped must be described by 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 lastmo
      count extents dsorg recfm1-recfm4 aloc blksize
      lrecl secaloc tt r tracks volume cchhr)
```

Code continued on next page...

Z/OS Options (continued)

SAS code (continued):

```
     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;
```

Code continued on next page...

Z/OS Options (continued)

SAS code (continued):

```
 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.
    ...
```

Power of the SAS Filename Statement

SAS FILENAME: more capabilities available than INFILE/FILE statements

- assign a fileref to a single file, several files, or directory
- assign fileref to directory or library so INFILE can read members using parentheses in dataset name

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.

Reading a “member” of a directory.

The following example reads a file which is a member of a Windows directory…

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

Power of the SAS Filename Statement (continued)

Assign a fileref to a directory to allow data step functions to open the directory, find number of members, member names and more.

The following program uses the following functions:

- DOPEN: open directory
- DNUM: determine number of members
- DREAD: return file name of each member
- PATHNAME: return path of directory.

Use FILEVAR= and FILENAME= INFILE options, as well, to read all the members in a directory in a different way than we saw earlier.

Power of the SAS Filename Statement (continued)

```
%let mfileref=temp;
filename &mfileref 'c:\temp';
/*****************************************************/
/* Loop thru entries in dir and read each file found.  */
/* ************************************************** */
data x;
 did=dopen("&mfileref");                /* try to open dir  */
 if did = 0 then                        /* if error         */
  putlog "ERROR: Directory not          /* opening file     */
   found for mfilenames &mfileref";/*                      */
 else                                   /* if dir opened ok */
  do;                                   /* do block         */
   memberCount = dnum( did );           /* get member nbr   */
    do j = 1 to memberCount;            /* for each member  */
     fileName = dread( did, j );        /* in directory     */
     do;                                /* do this          */
```

Code continued on next page...

Power of the SAS Filename Statement (continued)

```
    pathname=pathname("&mfileref"); /* for this program   */
    flow=filename;                  /* grab flows         */
    path_flow=trim(pathname)!!'\'!!filename;
    infile dummyf filevar=path_flow /* point to file      */
                  filename=myinfile /* give filename      */
              end=eof lrecl=80 pad; /* end of file, lrecl */
    do until(eof);                  /* loop thru recs     */
     input name $ jansale           /* read variables     */
                  febsale marsale   /* more               */
                  savefile=myinfile /* save in normal var */
                  ;                 /* end of input       */
     output;                        /* output to file     */
    end;                            /* end read pgm lines */
   end;                             /* end if index >0    */
  end;                              /* end loop thru pgms */
  did = close(did);                 /* close program dir  */
 end;                               /* if dir opened      */
stop;                               /* stop data step     */
run;                                /* end of step        */
```

Power of the SAS Filename Statement (continued)

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.
```

Power of the SAS Filename Statement (continued)

Partial SAS Log (continued):

```
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

Read and write data to any authorized FTP server using FTP options in FILENAME statement.

- only base SAS and FTP required, no extra products
- programs flexible, but transfer time needed to move data from 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;
proc print data=mydata;         /* Print the data     */
run;
```

Reading From an URL

FILENAME can point to a web page's data

- effectively opens program to millions of online files

Example:

- read first 15 records from URL file
- write to SAS log with 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

FILE statement

- opposite of INFILE statement (in most respects)
- function: define output raw file

Many options are the same for FILE and INFILE statements. Some options are different, such as those to…

- define reports in SAS with unique options
- route reports to different destinations with ODS

Other attributes include…

- PUT statement writing to SAS log (by default)
- routing output to same or different external file using FILE statement
- indicating use of carriage control characters in file
- using statement in conditional (IF-THEN) processing because it is executable
- using multiple statements to write to more than one external file in a single DATA step.

Basic File Syntax

file-specification: identifies external file used to write output from PUT

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

File-specification Forms:

'external-file'

- specifies physical name of external file
- enclosed in quotation marks
- how operating environment recognizes file

Fileref

- specifies fileref of external file
- must be previously associated with external file in a FILENAME statement, FILENAME function, or a system command

fileref(file)

- specifies fileref of aggregate storage location and name of file/member residing in location

LOG

- reserved fileref directing output of PUT statement to SAS log.

PRINT

- reserved fileref, directs output of PUT statement to file as output is produced

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

Writing to the SAS Log

The SAS log contains SAS notes and other messages along with output.

- default output file is the SAS log
- PUTLOG writes to log
- PUT writes to most recently executed FILE statement

The report may not be formatted the way you want it, but it's 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;
```

Writing to the SAS Log (continued)

Partial SAS 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
. . .
```

The SAS Print File

FILE PRINT directs PUT to write to the SAS print file, using TITLES, DATE, and NUM options at the top of each page. As seen in the example below, FILE must precede PUT…

```
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 SAS Print File (continued)

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

Output files require more options and information as they became more complex, just as input files.

- many options same as INFILE options
- available in all operating environments
- operating system specific options available
- several options exclusively for report writing because FILE statement can produce printed report

In the SAS documentation below, there is a complete list of FILE options.

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;
```

Enhancing the Report (continued)

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
. . .					

Hey Certified SAS Developers, the Jig is Up!

Counting Remaining Lines on 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.

Use variables for line and column pointers to fill several file buffers and print when completely filled in.

Example:

```
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;
```

Code continued on next page...

A Full Page Report (continued)

SAS Code (continued):

```
 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;
```

A Full Page Report (continued)

Division H

name	div	sales
BETH	H	4822.12
CHRIS	H	233.11
JOHN	H	678.43
MARK	H	298.12
PAT	H	4009.21
STEVE	H	6153.32
WILLIAM	H	3231.75

Division S

name	div	sales
ANDREW	S	1762.11
BENJAMIN	S	201.11
JANET	S	98.11
JENNIFER	S	542.11
JOY	S	2442.22
MARY	S	5691.78
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.

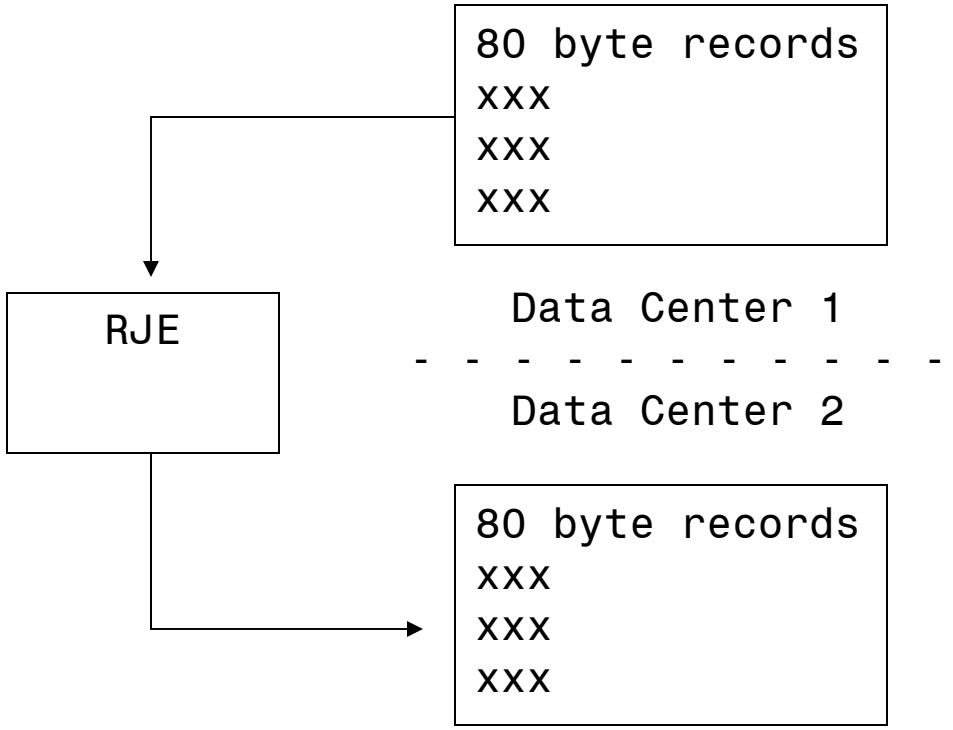
- use in any SAS DATA step needing to pass on raw data.

The rich DATA step programming language along with FILE options can serve as an extremely versatile utility program.

Several examples of writing raw files with FILE and PUT follows…

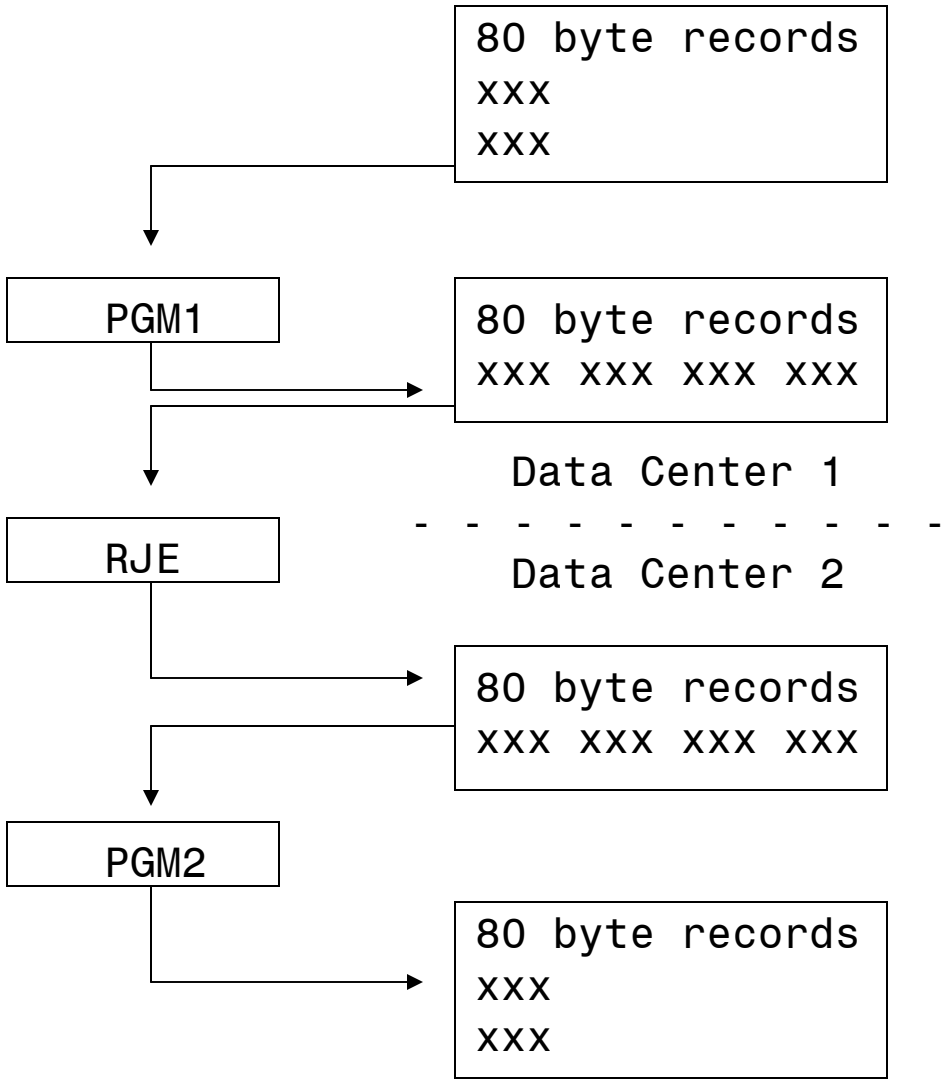
My First SAS Job

A data center transfers thousands of mostly blank records down a communications line. It takes a long time to move the data.



A SAS Solution

Write two SAS programs to "pack" and "unpack" data.



My First SAS Programs

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

```
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       */
```

Note:

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

A Copying Program

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;                                  /* end of step         */
```

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 to Every Letter

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

Many options available on INFILE are also available with FILE

These include:

- accessing the buffers
- altering recfm and termstr
- changing the output file dynamically
- encoding file
- writing binary file

FILENAME statement

- lets us write to email, FTP sites, aggregate file locations, and much more…
- usage with FILE statement as valuable for files that we need to create as they were with INFILE

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.

Conclusion (continued)

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 and Elizabeth First for advice and support in the preparation of this document

My Brain Is Full

"Mr. Osborne, may I be excused? My brain is full."

Contact Us

To subscribe to the free “Missing Semicolon” newslettter.

SYSTEMS SEMINAR CONSULTANTS, INC.

SAS® Training, Consulting, & Help Desk Services

Steven J. First

President

(608) 278-9964

FAX (608) 278-0065

sfirst@sys-seminar.com

www.sys-seminar.com

2997 Yarmouth Greenway Drive • Madison, WI 53711