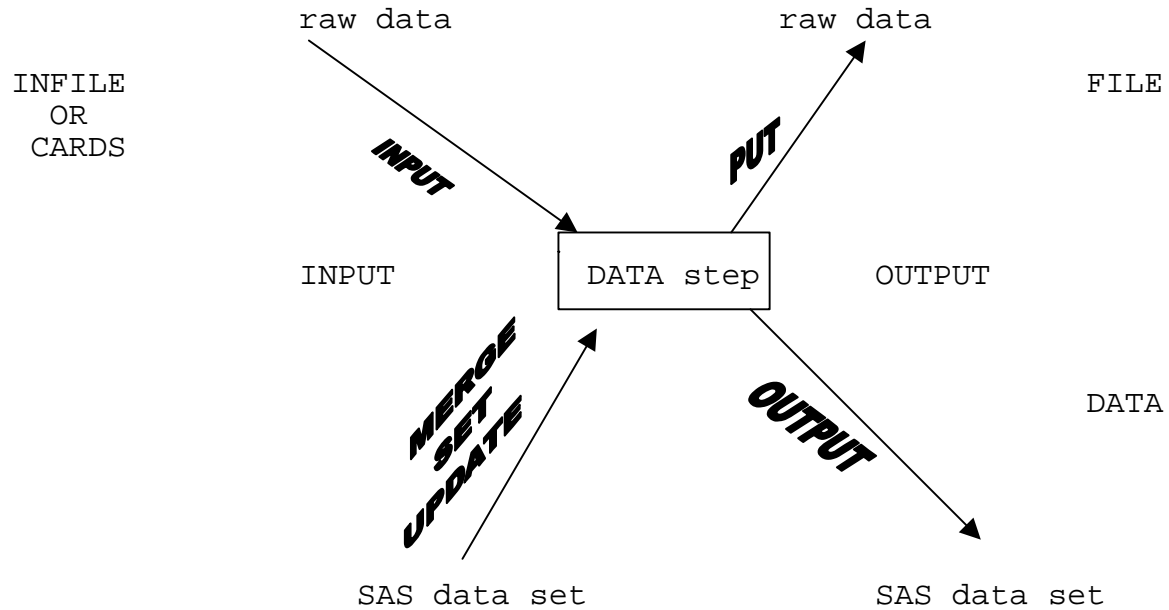


Introduction

Thus far, all the DATA step programs we have seen have involved reading and writing only SAS data sets. In this chapter we will present techniques to read and write external or "raw" files in the DATA step using the INPUT and PUT statements. Any combination of these inputs and outputs can be used in a DATA step.



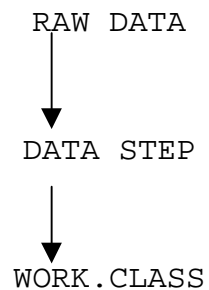
A. Reading External Files with SAS

The following example creates a temporary SAS data set from the raw data file CLASS.ASC

RAW DATA FILE CLASS.ASC

CHRISTIANSEN	M	37	71	195
HOSKING J	M	31	70	160
HELMS R	M	41	74	195
PIGGY M	F	.	48	.
FROG K	M	3	12	1
GONZO		14	25	45

```
FILENAME IN 'D:\BIOS111\CLASS.ASC';  
  
DATA CLASS ;  
  INFILE IN ;  
  INPUT NAME $ 1-12 SEX $ 15 AGE 18-19 HT 22-23 WT 26-28 ;  
RUN;  
  
PROC PRINT DATA=CLASS;  
RUN;
```



```

25  FILENAME IN 'D:\BIOS111\CLASS.ASC';
26
27  DATA CLASS ;
28  INFILE IN ;
29  INPUT NAME $ 1-12 SEX $ 15 AGE 18-19 HT 22-23 WT 26-28 ;
30  RUN;

```

NOTE: The infile IN is:
 FILENAME=D:\BIOS111\CLASS.ASC,
 RECFM=V,LRECL=132

NOTE: 6 records were read from the infile IN.
 The minimum record length was 28.
 The maximum record length was 28.

NOTE: The data set WORK.CLASS has 6 observations and 5 variables.

NOTE: The DATA statement used 1.0 seconds.

```

31  PROC PRINT DATA=CLASS;
32  RUN;

```

NOTE: The PROCEDURE PRINT used 0.13 seconds.

The SAS System

OBS	NAME	SEX	AGE	HT	WT
1	CHRISTIANSEN	M	37	71	195
2	HOSKING J	M	31	70	160
3	HELMS R	M	41	74	195
4	PIGGY M	F	.	48	.
5	FROG K	M	3	12	1
6	GONZO		14	25	45

Compilation Phase

During the compilation of the DATA step, the INPUT statement:

- ❖ Creates an input buffer to provide temporary storage for the current input record being processed.
- ❖ Creates the PDV containing the variables listed in the INPUT statement.
- ❖ Generates the machine language instructions for reading the data values from the input buffer and converting them into the corresponding SAS variable values in the PDV.
- ❖ Notes
 - ❖ The length of the input buffer is determined by the LRECL of the file referenced by the INFILE statement.
 - ❖ INFILE and INPUT statements are used only to read non-SAS files. To read SAS files, use a SET, MERGE, or UPDATE statement.

Execution Phase

During the execution phase of the DATA step, the input statement:

- ❖ Reads the next record from the input file into the input buffer.
- ❖ Reads each data value from the appropriate field in the input buffer, converts it to the specified SAS variable, and stores the result in the PDV.
- ❖ Notes
 - ❖ The PDV is initialized to missing each time control returns to the top of the DATA step.
 - ❖ Once variable values have been read into the PDV, they can be modified with DATA step programming statements in the same manner as values read from an existing SAS data set.

THE CARDS/DATALINES STATEMENT

- ❖ The CARDS and DATALINES statements are interchangeable
- ❖ Used to mark the start of a group of "raw" data lines to be read by SAS.
- ❖ Must be placed at the end of the data step(immediately following the return statement).
- ❖ The CARDS statement is followed by the data lines, and then a line containing only a semicolon.
- ❖ The general form is:

DATA new;

INPUT ;

*

*

* (SAS statements)

*

*

OUTPUT ;

RETURN ;

CARDS ; /* or DATALINES */

*

* (data lines)

*

;

RUN;

EXAMPLE: THE CARDS STATEMENT

DATA new;

INPUT A B C \$;

D = A + B ;

OUTPUT ;

RETURN ;

CARDS ; /* or DATALINES */

2 5 M

3 6 F

7 9 F

;

RUN;

THE FILENAME STATEMENT

The FILENAME statement associates a "FILEREf" (a valid SAS name) with an external file or device. An external file is a non-SAS file created on the host operating system (such as WINDOWS) from which you need to read data, SAS programming statements, or to which you want to write output. Once you associate a "FILEREf" with an external file, the "FILEREf" can be used as a shorthand reference for that file in the SAS programming statements (INFILE, FILE, and %INCLUDE) and display manager commands (FILE and INCLUDE) that access external files. You can associate a "FILEREf" with a single external file or an aggregate storage location (such as a directory) that contains many external files.

SYNTAX:

```
FILENAME fileref 'external file' <host options> ;
```

fileref is any valid SAS name
'external file' is the physical name of an external file. You can associate a fileref with a single file or an aggregate location.

EXAMPLES:

Example 1: Associating a Fileref with a single external file

This example reads data from a file that has been associated with the fileref BIOS .

```
filename bios 'd:\bios111\class.asc';  
  
data one ;  
  infile bios ;  
  input x y z ;  
run;
```

Example 2: Associating a Fileref with an aggregate storage location

If you associate a fileref with an aggregate storage location, you then use the fileref, followed in parentheses by an individual filename, to read from or write to any of the individual external files stored there.

```
filename bios 'd:\bios111';  
  
data one ;  
  infile bios(class.asc) ;  
  input x y z ;  
run;
```

The Infile Statement

The INFILE statement connects an external, non-SAS file with a DATA step in order to read the raw data records with an INPUT statement.

SYNTAX:

INFILE FILESPEC options:

where FILESPEC identifies the source of input data records to be read. As can be seen in the User's Guide, many of the INFILE options are technical in nature and have to do with Operating Systems features and file organizations that are beyond the scope of this course.

FILESPEC can have the following forms:

'EXTERNAL-file'

specifies the physical name of an external file. It must be enclosed in quotes.

FILEREF

gives the FILEREF of an external file. The FILEREF must have been associated previously with an external file in a FILENAME statement.

Selected options of interest include:

- DCB, RECFM, LRECL, BLKSIZE to specify DCB information about the file
- END, EOF, EOV to tell when you are processing the last record
- FIRSTOBS, OBS to define the first and last input records to be read
- LINESIZE, N to specify the number of columns and lines in the input buffer
- COLUMN, LINE give the current location of the pointers in the input buffer
- MISSOVER, STOPOVER determine what to do if too few data values are found in an input record

NOTES:

- ❖ The INFILE statement must be executed before the INPUT statement that reads the raw data lines.
- ❖ More than one INFILE statement may be used in a single DATA step. INPUT statements will read from the file defined by the most recently executed INFILE statement.

The Input Statement

- ❖ The INPUT statement reads raw data lines from an external (non-SAS) file specified in the INFILE statement and converts the values of the fields into the corresponding values of SAS variables in the Program Data Vector.
- ❖ Tells SAS to read one or more lines from the data source specified in the CARDS or INFILE statement
- ❖ The INPUT statement works as follows:
 - ❖ the contents of the line are copied into the computer's memory
 - ❖ SAS moves to a column indicated by INPUT and reads to the end of the data field
 - ❖ SAS converts the data to a SAS numeric or character variable and stores the SAS-formatted data in memory
 - ❖ The process is repeated for each variable indicated in the INPUT statement
 - ❖ Once all variables are read, they are written to the data set specified in the DATA statement
 - ❖ INPUT initiates the reading of data; INFILE or CARDS just tells SAS where to look for the data
 - ❖ You do not have to read every field of raw data into your SAS data set
 - ❖ You can read the same file more than once
- ❖ The general form of this statement is

INPUT list of variable names and specifications;

The variable names are new SAS variables being created by the INPUT statement. The specifications define the location of the input fields and how the values are to be converted to SAS's internal representation. The form of the specifications will vary depending on the input mode.

Column Input Mode

The most straightforward and commonly used mode of input specification is column mode. It should be used when the data values are in the same fields on all the data lines and the data values are in standard numeric or character form.

SYNTAX :

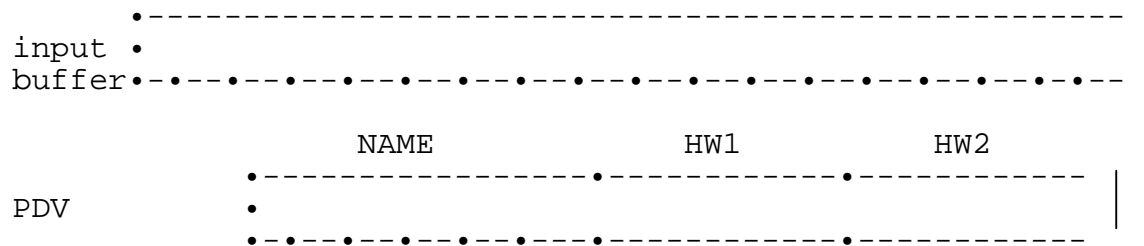
```
INPUT var1 start-end var2 $ start-end: . . ;
```

where:

var1 is a numeric variable starting in column "start" and ending in column "end", and
var2 \$ is a character variable starting in column "start" and ending in column "end."

Example: Create a temporary SAS data set from raw data

```
DATA STUDENT;
    INPUT NAME $ 1-6 HW1 10-11 HW2 15-17;
    CARDS;
JOHN B      9      90
ALBERT     10      95
SALLY      .        8
MARY       7      74
;
run;
PROC PRINT DATA=STUDENT;
    ID NAME;
    TITLE 'WORK.STUDENT' ;
```



WORK.STUDENT

NAME	HW1	HW2
JOHN B	9	90
ALBERT	10	95
SALLY	.	8
MARY	7	74

Notes on Column Input Mode

- ❖ The field positions for the variables are fixed
- ❖ A \$ is used to indicate character variables
- ❖ Data fields can be read and reread in any order.

```
INPUT HW1 10-11 NAME $ 1-6 HW2 15-17; INPUT NAME $ 1-6 HW1 10-11  
HW2 15-17 HW2CHAR $ 15-17;
```

- ❖ Blanks are read as missing
- ❖ For character variables:
 - ❖ embedded blanks are allowed in fields (“JOHN SMITH”)
 - ❖ data values are left justified
 - ❖ the length of the variable is determined by the column specification
 - ❖ character data values can range from 1-200 in length in releases of SAS prior to 7.0
 - ❖ character data values can range from 1-32,767 in length starting with Version 7.0 of SAS
- ❖ For numeric variables:
 - ❖ the value can be anywhere within the column specification
 - ❖ embedded blanks are not allowed in a numeric field

List Input Mode

In list input the variables are specified in the order in which they appear on the data lines. No field specification is given, but character variables are followed by a \$. A pointer is used to determine which bytes of the input buffer correspond to each variable in the INPUT statement. The pointer scans the input buffer until it encounters a non-blank character. That character and all consecutive non-blank characters on the line comprise the data value. The next blank signals the end of that data field and the process continues.

Syntax:

```
INPUT var . . . var $ . . . ;
```

Example

```
DATA CLASS;
  INFILE STUDENT;
  INPUT NAME $ SEX $ AGE HT WT;
  OUTPUT;
  RETURN;
RUN;
```

```
OS file  HOSKING M 31 70 160
         HELMS R  M 41 74 195
         CHRISTIANSEN M 37 71 185
         PIGGY  F  . 48  .
         FROG  F  3 12 1
         GONZO  . 14 25 45
```

```
input buffer  .-----|
              .
              .-----|
              .

              NAME          SEX          AGE          HT          WT
PDV           .-----|
              .
              .-----|
              . . . . .
```

WORK.CLASS

```
SAS
dataset      NAME          SEX          AGE          HT          WT
              HOSKING      M           31           70          160
              HELMS        R           .            41           74
              CHRISTIA     M           37           71          185
              PIGGY        F           .            48           .
              FROG         F           3            12           1
              GONZO        .           14           25           45
```

Notes on List Input Mode:

- ❖ List input is needed when data values are stored with one blank between each field instead of being stored in fixed fields
- ❖ Data lines can be free format
- ❖ A \$ is used to indicate character variables.
- ❖ Every field must be specified in order
- ❖ Data fields must be separated from each other by at least one blank.
- ❖ Character variables default to 8 bytes.
- ❖ Embedded blanks are not allowed in character variables.
- ❖ Missing values must be coded as "." (as a place holder) for numeric and character variables.
- ❖ Blank fields cause the matching of variable names and values to get out of sync.

Formatted Input Mode

The most flexible (and complex) input mode is formatted input. This mode specifies explicit pointer controls and informats for each variable in the INPUT statement.

An informat is a set of directions for converting the characters in an input field to a variable's value in one of SAS's two types of internal representation. The wide variety of informats available allows SAS to read data written in "virtually any form."

With formatted input you specify the starting location and field width, move an input "pointer" to the starting position of the field, then specify the variable and an informat.

SYNTAX:

```
INPUT pointer-control variable informat . . . ;
```

where pointer-control directs the pointer to the specified byte of input buffer and informat defines the input format and field width to be used to read the variable's value.

Pointer Controls

@n (absolute)	go to byte n of input buffer
+n (relative)	move pointer n bytes to the right
w (informat)	format width specification moves pointer to the right as it reads the data value

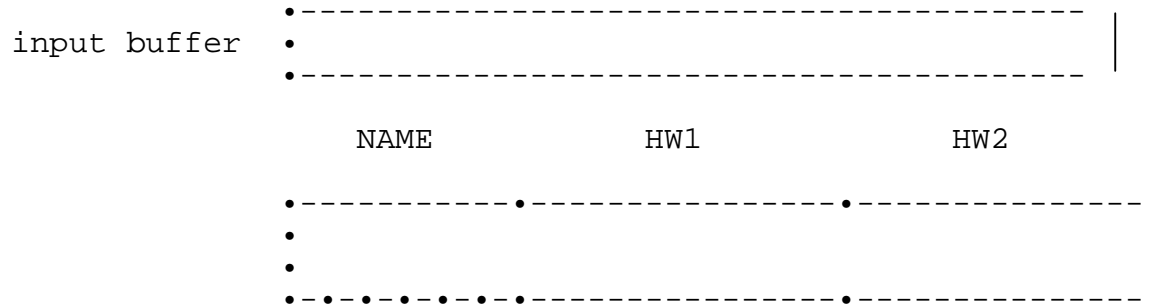
Selected Informats

w.	standard numeric
w.d	standard numeric with decimal
\$w.	standard character
\$CHARw.	characters with leading blanks
HEXw.	numeric hexadecimal
\$HEXw.	character hexadecimal
Ew.d	scientific notation
BZw.d	blanks are zeros
\$VARYINGw.length	self-defining string lengths
COMMAw.d	commas, dollar signs, and other Special characters in numbers and negative numbers in parentheses.

Example:

```
INPUT @1 NAME $6. +3 HW1 2. @15 HW2 3.;
```

JOHN B	9	90	raw data
ALBERT	10	95	
SALLY		8	
MARY	5	8	



WORK.STUDENT

NAME	HW1	HW2
JOHN B	9	90
ALBERT	10	95
SALLY	.	8
MARY	5	8

Formatted Input Example:

The postal codes and 1980 population for five states in the southeast are shown in the table below. Read the data and create a SAS data set.

F	L		9	,	7	3	9	,	9	9	2
G	A		5	,	4	6	4	,	2	6	5
N	C		5	,	8	7	4	,	4	2	9
S	C		3	,	1	1	9	,	2	0	8
V	A		5	,	3	4	6	,	2	7	9

```
Filename xyz 'c:\bios111\sepop.asc' ;  
Data pops;  
Infile xyz ;  
Input @1 input state $2. +1 pop comma9.;  
Run;
```

WORK.POPS

OBS	STATE	POP
1	FL	973999
2	GA	546426
3	NC	587442
4	SC	311920
5	VA	534627

Formatted Input/DATES Example:

```
DATA ONE ;  
  
INPUT @1 DATE1 MMDDYY6. @8 DATE2 6. @15 DATE3 DATE7. @15 DATE4 $7. ;  
  
CARDS;  
011593 011593 15JAN93  
;  
RUN;  
  
PROC PRINT;  
RUN;  
  
PROC CONTENTS;  
RUN;
```

FORMATTED INPUT/DATES EXAMPLE

OBS	DATE1	DATE2	DATE3	DATE4
1	12068	11593	12068	15JAN93

FORMATTED INPUT/DATES EXAMPLE

CONTENTS PROCEDURE

-----Alphabetic List of Variables and Attributes-----

#	Variable	Type	Len	Pos
1	DATE1	Num	8	0
2	DATE2	Num	8	8
3	DATE3	Num	8	16
4	DATE4	Char	7	24

Formatted Input Example:

```
DATA ONE ;
```

```
INPUT X Y ;
```

```
CARDS;
```

```
3 4
```

```
5 6
```

```
7 F
```

```
;
```

```
RUN;
```

```
15 DATA ONE ;
```

```
16
```

```
17 INPUT X Y ;
```

```
18
```

```
19 CARDS ;
```

NOTE: Invalid data for Y in line 22 3-3.

```
RULE:----+----1----+----2----+----3----+----4----+----5----+----6----+----7---
```

```
22 7 F
```

```
X=7 Y=. _ERROR_=1 _N_=3
```

NOTE: The data set WORK.ONE has 3 observations and 2 variables.

NOTE: The DATA statement used 0.53 seconds.

```
23 ;
```

```
24
```

```
25 RUN ;
```

TRALING @

An INPUT statement ending with "@" (before the ";") holds the current data line in the input buffer for the next INPUT statement to read. This is extremely useful when reading data files containing mixed record types. For example, consider records from two versions of a form in the same file.

```
DATA ONE;
```

```
    INPUT VERSION 1 @;
```

```
    IF VERSION EQ 1 THEN INPUT NAME $ 2-9 SEX $ 11 AGE 13-14;
```

```
    IF VERSION EQ 2 THEN INPUT NAME $ 2-9 SEX $10 AGE 12-13 HT 15-16;
```

```
    CARDS;
```

```
1HOSKING M 31
```

```
2HELMS  M 40 70
```

```
1CHRISTIA M 37
```

```
2PIGGY F 48
```

```
;  
RUN;
```

TRALING @ EXAMPLE

The raw data file LIPA.ASC contains data with multiple record types. Read in data where form=LIP and create a SAS data set.

```
FILENAME RAW 'C:\BIOS111\LIPA.ASC';
```

```
DATA ONE ;
```

```
    INFILE RAW ;
```

```
    INPUT FORM $ 1-3 @ ;
```

```
    IF FORM='LIP' THEN DO;
```

```
        INPUT X 4-5 Y 6-7 Z 8-9 ;
```

```
        OUTPUT ;
```

```
    END;
```

```
RUN;
```

Format Lists

A variation of formatted input is called "format list mode," in which a list of variable names and a list of pointer controls and formats are grouped separately by parentheses:

```
INPUT (NAME SEX AGE HT WT)
      ($8. @11 $1 2. +1 4. @21 5.);
```

This is really only an advantage when there are a series of variables with a repeating format.

For example,

```
INPUT X1 8. X2 8. X3 8. X4 8. X5 8.;
```

can be rewritten as:

```
INPUT (X1-X5) (5*8.);
```

The format list cycles to satisfy the variable list:

```
INPUT (X1-X5) (8.);
```

```
INPUT (X1 Y1 X2 Y2 X3 Y3) (2. 3.);
```

```
INPUT (X1-X2) (2. +3) @1 (Y1-Y3) (+2 3.);
```

In general, this mode of input is the most error prone; avoid it when possible.

Mixing Input Styles

You can mix the three INPUT styles (list, column, and formatted) in one INPUT statement.

```
INPUT NAME $ 11 SEX $1. AGE 13-14 HT @21 WT 5.;
```

Named Input Mode

Named input is a special input mode that allows the data lines to contain the variable name as well as its value.

SYNTAX:

```
INPUT var1 ... namedvar1= namedvar2= ... ;
```

Data lines can then specify the values of the named variables in any order as long as named variables are after regular input fields. This mode is useful in constructing transaction data sets for subsequent use with the UPDATE statement.

Example

```
DATA TRANS;  
    INFILE FIXES  
    INPUT ID AGE= WT= HT= ;  
RUN;
```

OS file

```
243 AGE=12 WT=99 HT=50  
475 WT=120 AGE=11 HT=45  
649 HT=60
```

WORK.TRANS

ID	AGE	WT	HT
243	12	99	50
475	11	120	45
649	.	.	60

Multiple Records Per Observation

There are several ways to read an observation that continues over multiple input records. Consider input data consisting of 3 lines per person.

- ❖ Use multiple input statements to advance to the next input line. Each input statement advances to the next record.

```
DATA CLASS;
    INFILE RAWDATA;
    INPUT NAME $ 1-8 SEX $ 11
    INPUT AGE;
    INPUT HT 1-7 WT 8-14;
```

- ❖ Use / to advance to the next input line. The / advances the pointer to the next line of raw data.

```
INPUT NAME @ 1-8 SEX $11 / AGE / HT 1-7 WT 8-14;
```

RAWDATA

HOSKING	M
31	
70	160

input
buffer

•-----
•
•-----

NAME SEX AGE HT WT

PDV

--	--	--	--	--

Multiple Line Input Buffers: The Line Pointer

SAS can be instructed to set up a multiple-line input buffer. A line pointer is moved from line to line to control input processing. Use #n to advance to the first column of the nth record in the group (i.e., to set the line pointer to the nth line in the input buffer):

```
INPUT #1 NAME $ 1-8 SEX $ 11
      #2 AGE 1-2
      #3 HT 1-7 WT 8-14;
```

or another order:

```
INPUT #3 HT 1-7 WT 8-14
      #1 NAME $ 1-8 SEX $11
      #2 AGE 1-2;
```

The highest #n will be the number of lines in the input buffer. In this example, the input buffer has 3 lines:

HOSKING	M
31	
70	160

	NAME	SEX	AGE	HT	WT
PDV					

The infile statement option N= can also be used to declare the number of lines in the buffer.

Summary: Pointer Controls

	column	line
absolute	@n	#n
relative	+n	/

Notes:

- ❖ n can be a positive numeric constant or a numeric variable
- ❖ For the relative column shift (+n), n can also be a numeric variable with a negative value

The Input Function

The INPUT function is similar to the INPUT statement in that it reads a string of characters and converts them to a SAS variable value according to a specified format. It differs from the INPUT statement in that it "reads" the string of characters from a SAS character variable rather than from an external file.

SYNTAX:

variable = INPUT (charactervariable, format);

The length and type (character or numeric) of "variable" will be determined by "format".

Input function Example

```
DATA ONE ;
  INPUT @1 VAR1 $2. @4 DATE1 $6. ;

  NVAR1 = INPUT(VAR1,2.) ;

  NDATE1 = INPUT(DATE1,MMDDYY.) ;

  CARDS;
10 011593
20 103193
;
RUN;

TITLE 'INPUT EXAMPLE';
PROC CONTENTS DATA=ONE;
RUN;
PROC PRINT DATA=ONE;
FORMAT NDATE1 MMDDYY6. ;
RUN;
```

-----Alphabetic List of Variables and Attributes-----

#	Variable	Type	Len	Pos	Label
2	DATE1	Char	6	6	
4	NDATE1	Num	8	20	
3	NVAR1	Num	8	12	
1	VAR1	Char	2	4	

INPUT EXAMPLE

OBS	VAR1	DATE1	NVAR1	NDATE1
1	10	011593	10	011593
2	20	103193	20	103193

Writing External Files With SAS

In the previous sections raw data records were read into the DATA step using INFILE and INPUT statements. This section will present the opposite case; existing SAS variables will be written to an external file using the FILE and PUT statements. The following example reads an existing SAS data set and writes raw data to an external file.

```
FILENAME RAW 'C:\BIOS111\CLASS.RAW' ;
```

```
DATA TEMP;  
  SET CLASSLIB.CLASS;  
  FILE RAW;  
  PUT NAME $ 1-8 AGE 10-12 HT 15-20;  
RUN;
```

class

OBS	NAME	SEX	AGE	HT	WT
1	CHRISTIANSEN	M	37	71	195
2	HOSKING J	M	31	70	160
3	HELMS R	M	41	74	195
4	PIGGY M	F	.	48	.
5	FROG K	M	3	12	1
6	GONZO		14	25	45

DATA
STEP

--

RAW

CHRISTIANSEN	37	71.0
HOSKING J	31	70.0
HELMS R	41	74.0
PIGGY M	.	48.0
FROG K	3	12.0
GONZO	14	25.0

THE FILE STATEMENT

FILE *file-specification* <options> <host options>

- ❖ The FILE statement specifies the output file for PUT statements in the current data step
- ❖ By default PUT statement output is written to the SAS log
- ❖ The FILE statement specifies an external file, not a SAS data set
- ❖ More than one FILE statement can be used in a data step. PUT statements will write to the file defined by the most recently executed FILE statement.
- ❖ The FILE statement can be executed conditionally
- ❖ File specification

identifies an external file to which you want to write output with a PUT statement. File specification can have the following forms:

'external-file'

specifies the physical name of the external file and must be enclosed in quotes.

fileref

gives the fileref of an external file. The fileref must have been previously associated with an external file in a FILENAME statement.

LOG

directs lines produced by PUT statements to the SAS log. PUT statements are by default written to the LOG. To write PUT statements to the LOG a FILE statement is not needed. A FILE LOG statement is only used to restore the default action or to specify additional options.

PRINT

directs lines produced by PUT statements to the same print file as the output produced by SAS procedures. When PRINT is the fileref, SAS uses carriage control characters and writes the lines with the characteristics of a print file.

❖ Selected Options

DCB, RECFM, LRECL, BLKSIZE	to specify DCB information about the file
HEADER, NOTITLE	to have user supplied titles for external print files.
COLUMN, LINE	give the current locations of the pointer in the output buffer.
PAGESIZE, LINESIZE, N	to specify the number of lines per page and the number of columns and lines in the output buffer.

THE PUT STATEMENT

The PUT statement is almost exactly the opposite of the INPUT statement. The PUT statement writes the values of SAS variables to the output buffer, which is then written to the external file defined by the FILE statement.

SYNTAX:

PUT <specification1> <specification2> .. ;

Where specification can have any of the following forms:

Column	variable start-end decimal
List	variable
Format	variable format
Named	variable =
Pointer-control	@n, +n, /, #n, @
<u>_INFILE_</u>	writes contents of the input buffer to the output buffer
<u>_ALL_</u>	writes the values of all variables in the PDV using named output
<u>_PAGE_</u>	advances to the top of a new page in the output file (this options is only available for print files)

- ❖ Column, list, formatted, named modes, and format lists are the same as defined in the INPUT statement.
- ❖ _INFILE_ and _ALL_ are useful for debugging your programs.
- ❖ PUT statement writes lines to:
 - ❖ the SAS log
 - ❖ the SAS procedures output file
 - ❖ an external file
 - ❖ to the location specified in the most recently executed FILE statement
- ❖ If no FILE statements are executed before a PUT statement, lines are written to the SAS log

- ❖ The PUT statement can write lines containing:
 - ❖ variable values
 - ❖ character strings
 - ❖ a combination of the above
- ❖ With specifications you can list items to be written, indicate their position, and specify how they are to be formatted.
- ❖ You can write variable values in LIST, COLUMN, FORMATTED, or NAMED output mode.

COLUMN OUTPUT

```
put name 1-20 var1 22-23 var2 25-28 ;
```

LIST OUTPUT

```
put name var1 var2 ;
```

FORMATTED OUTPUT

```
PUT @1 NAME $CHAR20. @22 VAR1 2. @25 VAR2 $3. ;
```

NAMED OUTPUT

```
PUT NAME= VAR1= VAR2= ;
```

Note:

If a data step exists for the sole purpose of writing values with PUT statements, the special data set name, `_NULL_`, can be used on the DATA statement to prevent a SAS data set from being created.

PUT EXAMPLE

```
1 DATA ONE ;  
2  
3 SET SC.CLASS ;  
4  
5  
6 PUT NAME SEX HT ;  
7 PUT NAME= SEX= HT= ;  
8  
9 RUN;
```

```
CHRISTIANSEN M 71  
NAME=CHRISTIANSEN SEX=M HT=71  
HOSKING J M 70  
NAME=HOSKING J SEX=M HT=70  
HELMS R M 74  
NAME=HELMS R SEX=M HT=74  
PIGGY M F 48  
NAME=PIGGY M SEX=F HT=48  
FROG K M 12  
NAME=FROG K SEX=M HT=12  
GONZO 25  
NAME=GONZO SEX= HT=25
```

NOTE: The data set WORK.ONE has 6 observations and 5 variables.

NOTE: The DATA statement used 4.87 seconds.

PUT EXAMPLE

```
10 DATA ONE ;
11
12 SET SC.CLASS ;
13
14
15 IF SEX='M' THEN PUT 'DATA DUMP: ' _ALL_ ;
16
17
18 RUN;
```

DATA DUMP: NAME=CHRISTIANSEN SEX=M AGE=37 HT=71 WT=195 _ERROR_=0
N=1

DATA DUMP: NAME=HOSKING J SEX=M AGE=31 HT=70 WT=160 _ERROR_=0 _N_=2

DATA DUMP: NAME=HELMS R SEX=M AGE=41 HT=74 WT=195 _ERROR_=0 _N_=3

DATA DUMP: NAME=FROG K SEX=M AGE=3 HT=12 WT=1 _ERROR_=0 _N_=5

NOTE: The data set WORK.ONE has 6 observations and 5 variables.

NOTE: The DATA statement used 8.25 seconds.

PUT EXAMPLE

```
1 DATA _NULL_ ;
2 SET SC.CLASS ;
3 IF SEX NOT IN('M','F') THEN PUT
4 'BAD VALUE FOR SEX ' NAME= +2 SEX= ;
5 IF AGE <=.Z THEN PUT NAME +3 'ISN'T TELLING';
6 IF HT <20 THEN PUT NAME= HT= 8.2 ;
7 RUN;
```

PIGGY M ISN'T TELLING

NAME=FROG K HT=12.00

BAD VALUE FOR SEX NAME=GONZO SEX=

NOTE: The DATA statement used 0.39 seconds.

PUT EXAMPLE

```
24 DATA ONE ;
25
26 SET SC.CLASS ;
27
28 IF (SEX NE 'F') & (SEX NE 'M') THEN DO;
29
30     PUT 'INVALID VALUE FOR SEX ' NAME SEX ;
31
32     DELETE ;
33 END;
34 RUN;
```

INVALID VALUE FOR SEX GONZO

NOTE: The data set WORK.ONE has 5 observations and 5 variables.

NOTE: The DATA statement used 0.5 seconds.

PUT EXAMPLE: CHECKING INPUT DATA

```
99 DATA ONE ;
100
101 INPUT ID SEX MS $ SS ;
102
103 IF NOT (1<=SEX<=2) THEN PUT
104     'INVALID VALUE FOR SEX ' +2 ID= +2 SEX= ;
105
106 IF NOT (MS IN('M','N','O')) THEN PUT
107     'INVALID VALUE FOR MS' +2 ID= +2 MS= ;
108
109
110 CARDS ;
```

INVALID VALUE FOR SEX ID=1 SEX=0

INVALID VALUE FOR MS ID=3 MS=Q

NOTE: The data set WORK.ONE has 4 observations and 4 variables.

NOTE: The DATA statement used 0.4 seconds.

PUT EXAMPLE: CREATING AN EXTERNAL FILE/DATA _NULL_

```
172 FILENAME RAW 'D:\BIOS111\PUTEX.DAT';
173
174 DATA _NULL_ ;
175
176 SET SC.CLASS ;
177
178 FILE RAW ;
179
180 PUT @1 NAME $CHAR20. @22 SEX $1. @24 AGE 3.
181     @28 HT 3. @32 WT Z3. ;
182
183 RUN;
```

NOTE: The file RAW is:

```
FILENAME=D:\BIOS111\PUTEX.DAT,
RECFM=V,LRECL=132
```

NOTE: 6 records were written to the file RAW.

The minimum record length was 34.

The maximum record length was 34.

NOTE: The DATA statement used 0.71 seconds.

PUT STATEMENT EXAMPLE: CHECKING INPUT DATA

```
39 data _null_ ;
40 set sc.v1(in=in1)
41     sc.v2(in=in2) end=eof ;
42 by id;
43
44 count1 + in1 ;
45 count2 + in2 ;
46 count3 + first.id ;
47
48 if (eof) then put
49
50     '*****' /
51     ' # of records in v1 is: ' count1 /
52     ' # of records in v2 is: ' count2 /
53     ' # of unique records is: ' count3 /
54     '*****' ;
55
56
57 run;
```

```
*****
```

```
# of records in v1 is: 151
```

```
# of records in v2 is: 148
```

```
# of unique records is: 151
```

```
*****
```

NOTE: The DATA statement used 0.42 seconds.

PUT STATEMENT EXAMPLE: CORRECTING DATA

```
14 DATA ONE;
15
16 SET SC.CLASS ;
17
18
19 IF SEX=' ' THEN DO;
20
21     PUT 'BEFORE CORRECTED: ' NAME= SEX= ;
22
23     SEX='F' ;
24
25
26     PUT 'AFTER CORRECTED: ' NAME= SEX= ;
27 END;
28
29 RUN;
```

BEFORE CORRECTED: NAME=GONZO SEX=

AFTER CORRECTED: NAME=GONZO SEX=F

NOTE: The data set WORK.ONE has 6 observations and 5 variables.

NOTE: The DATA statement used 1.53 seconds.

PUT FUNCTION

`PUT(source,format) ;`

- ❖ Writes the value of a variable to a character variable using the specified format
- ❖ The result of the PUT function is always a character string
- ❖ The format must be the same type as the source
- ❖ If the source is numeric the resulting string is right aligned
- ❖ If the source is character the resulting string is left aligned
- ❖ The PUT function can be used to convert a numeric variable to a character variable
- ❖ The PUT function is also an efficient way to recode a variable
- ❖ To preserve the result of the PUT function you must assign it to a new variable

- ❖ Examples:

```
x=44 ;  
y=put(x,4.) ; ** y=' 44' ** ;
```

```
a=123;  
b=put(a,z4.) ; ** b='0123' ** ;
```

PUT FUNCTION EXAMPLE: RECODING

Recode age to a character variable with values group1, group2, or invalid.

```
184 PROC FORMAT ;
185   VALUE AFT
186   0-40   = 'GROUP 1'
187   40<-HIGH = 'GROUP 2'
188   OTHER   = 'INVALID' ;
NOTE: Format AFT has been output.
189 RUN;
```

NOTE: The PROCEDURE FORMAT used 5.05 seconds.

```
190
191
192 DATA ONE ;
193   SET SC.CLASS(KEEP=NAME AGE) ;
194
195   NEWAGE=PUT(AGE,AFT7.) ;
196
197   PUT _ALL_ ;
198 RUN;
```

```
NAME=CHRISTIANSEN AGE=37 NEWAGE=GROUP 1 _ERROR_=0 _N_=1
```

```
NAME=HOSKING J AGE=31 NEWAGE=GROUP 1 _ERROR_=0 _N_=2
```

```
NAME=HELMS R AGE=41 NEWAGE=GROUP 2 _ERROR_=0 _N_=3
```

```
NAME=PIGGY M AGE=. NEWAGE=INVALID _ERROR_=0 _N_=4
```

```
NAME=FROG K AGE=3 NEWAGE=GROUP 1 _ERROR_=0 _N_=5
```

```
NAME=GONZO AGE=14 NEWAGE=GROUP 1 _ERROR_=0 _N_=6
```

NOTE: The data set WORK.ONE has 6 observations and 3 variables.

NOTE: The DATA statement used 4.62 seconds.

PUT FUNCTION EXAMPLE: RECODING

Recode age to a numeric variable with values 1,2,3 or missing.

```
226 PROC FORMAT ;
227   VALUE AFT
228     0-30   = '1'
229     30<-40 = '2'
230     40<-HIGH = '3'
231     OTHER  = '.';
```

NOTE: Format AFT has been output.

```
232 RUN;
```

NOTE: The PROCEDURE FORMAT used 1.04 seconds.

```
233
```

```
234
```

```
235 DATA ONE ;
```

```
236   SET SC.CLASS(KEEP=NAME AGE) ;
```

```
237
```

```
238   TEMP=PUT(AGE,AFT1.) ;
```

```
239   NEWAGE=INPUT(TEMP,1.) ;
```

```
240
```

```
241   /* OR
```

```
242     NEWAGE=INPUT((PUT(AGE,AFT1.)),1.) ;
```

```
243   */
```

```
244
```

```
245   PUT _ALL_ ;
```

```
246 RUN;
```

```
NAME=CHRISTIANSEN AGE=37 TEMP=2 NEWAGE=2 _ERROR_=0 _N_=1
```

```
NAME=HOSKING J AGE=31 TEMP=2 NEWAGE=2 _ERROR_=0 _N_=2
```

```
NAME=HELMS R AGE=41 TEMP=3 NEWAGE=3 _ERROR_=0 _N_=3
```

```
NAME=PIGGY M AGE=. TEMP=. NEWAGE=. _ERROR_=0 _N_=4
```

```
NAME=FROG K AGE=3 TEMP=1 NEWAGE=1 _ERROR_=0 _N_=5
```

```
NAME=GONZO AGE=14 TEMP=1 NEWAGE=1 _ERROR_=0 _N_=6
```

NOTE: The data set WORK.ONE has 6 observations and 4 variables.

NOTE: The DATA statement used 3.72 seconds.

Converting Databases to SAS: DBMS/COPY

A. What is DBMS/COPY ?

software that translates data from one software package to another

DBMS/COPY can be run in interactive or batch mode

B. Starting DBMS/COPY - Interactive Mode

click on **DBMSCOPY** icon or click on **DBMSWIN.EXE** from file manager

DBMS/COPY can also be run under DOS

C. Input selection for a Dbase IV/Foxpro file

from main menu select **interactives**

select **copy database**

from **list files of type** box select :

dBase 4

select input drive and directory

select input file to convert from **File Name** box

click on OK

click on OK from **Power Panel** Box

D. Output selection for a SAS file

from **list files of type** box select :

SAS for Windows 6.08 or 6.10(*.SD2) or

SAS for PC/DOS 6.04 (*.SSD) or

SAS Transport V6

select input drive and directory

in the **File Name** box type in name of New SAS data set to be created

click on OK

E. To convert Dbase IV/Foxpro file to SAS File

DBMS/COPY will ask you to verify the input and output file names. Click on **DO-It!** if they are correct and the transfer will begin.

F. To convert in Batch Mode

create a file with the following statments

compute:

in=c:\demo\eba.dbf

out=c:\demo\eba.sd2 ;

run;

Converting Databases to SAS: SAS/ACCESS

A. What is SAS/ACCESS SOFTWARE

software that provides an interface between SAS and several database products such as DBASE, FOXPRO, and EXCEL

to use SAS/ACCESS you must be running SAS Version 6.06 or greater

B. What can SAS/ACCESS be used for ?

- create SAS/ACCESS descriptor files
- access data in a database product's tables (such as FOXPRO or DBASE IV)
- extract a database product's data and place them in a SAS data file

C. SAS/ACCESS: Overview

- the Access procedure allows you to describe a database file(or table) to SAS
- the description is stored in SAS/ACCESS descriptor files
- descriptor files can be used in SAS programs much the same way that you would use SAS data sets
- you can print, plot, and model data described by the descriptor files
- you can use descriptor files to create SAS data sets

D. SAS/ACCESS Descriptor Files

- used to establish a connection between SAS and a database product
- are created using an interactive windowing procedure or in batch mode using the ACCESS procedure
- once created, descriptor files can be used in non-interactive mode
- there are 2 types of descriptor files: access descriptors and view descriptors

E. Access Descriptor Files

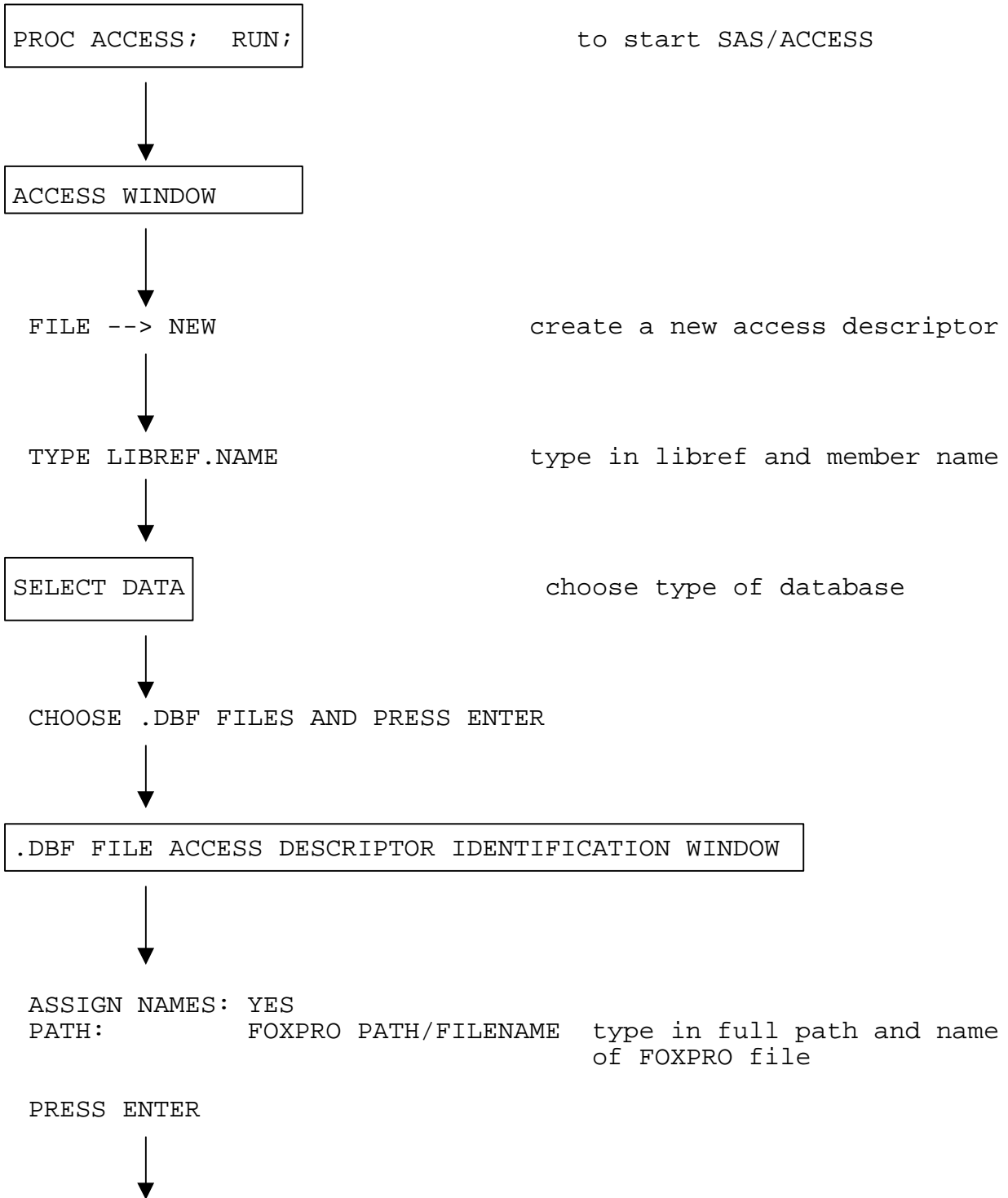
- are of member type access(.SA2)
- describes only one database file
- holds essential information about a database file such as the name of the database, the pathname, database field names and types
- contains corresponding SAS information such as default SAS variable names and formats

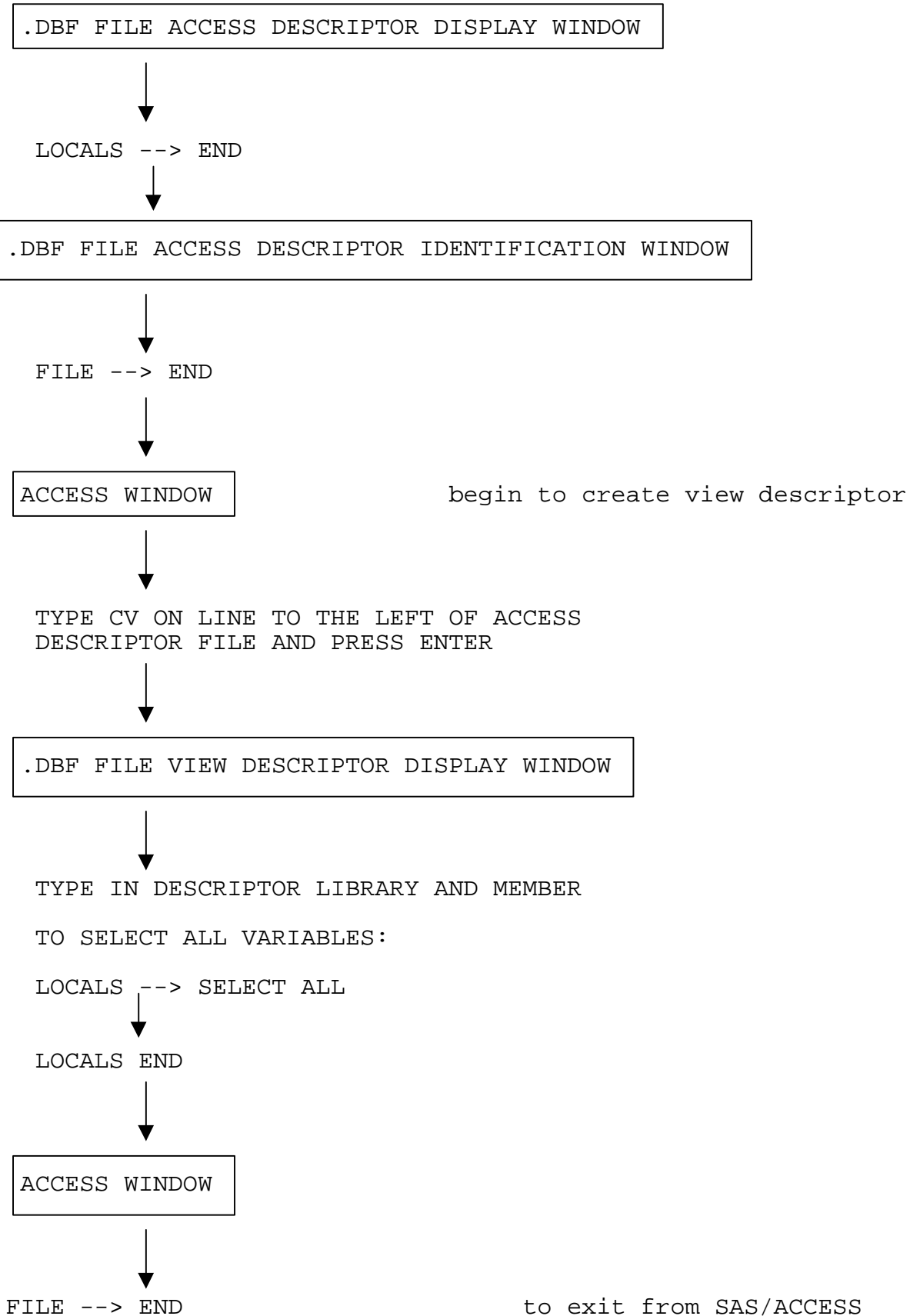
F. View Descriptor Files

- are of member type view(.SV2)
- can define all of the data or a subset of the data described by one access descriptor
- can be used to select desired rows and columns

ACCESS AND VIEW DESCRIPTOR FILES INTERACTIVE MODE

FROM PROGRAM WINDOW SUBMIT:





USING VIEW DESCRIPTOR FILES

A. Creating a SAS data set from an access view descriptor file

```
PROC ACCESS VIEWDESC = LIBREF.VIEWNAME  
    OUT = LIBREF.SASNAME ;
```

B. Using access view descriptor files

- views can be used similar to SAS data sets for most applications
- for example if we created 2 views in PROC ACCESS called ADA & AVA, we can do the following:

```
LIBNAME SC V612 'D:\DAIS\SASTEST';
```

```
DATA ONE ;  
    SET SC.ADA ;
```

other sas statements

```
RUN;
```

```
DATA ONE ;  
    SET SC.ADA SC.AVA ;  
    WHERE DATATYPE ='D' ;
```

other sas statements

```
RUN;
```

```
PROC PRINT DATA=SC.ADA(OBS=5);  
RUN;
```

```
PROC MEANS DATA=SC.ADA ;  
RUN;
```

CREATING ACCESS VIEW DESCRIPTOR FILES: BATCH MODE

Example:

```
proc access dbms=dbf ;    ** dbms can also equal dif **;  
  
create sc.cxi.access ;    ** create access descriptor **;  
  
path='c:\dais.dbf' ;    ** location of dbase file **;  
  
assign=yes ;            ** generate unique SAS variable names ** ;  
  
rename visit=visno;    ** rename SAS variable ** ;  
  
format visit $char2. ; ** assign SAS format ** ;  
  
list all ;              ** list columns in the descriptor ** ;  
  
create sc.cxi.view ;    ** create view descriptor ** ;  
  
select all ;           ** select all columns ** ;  
  
subset where delete_f is missing ; * specify selection criteria *;  
  
run;
```

Converting Databases to SAS: DYNAMIC DATA EXCHANGE(DDE)

DDE is a method of dynamically exchanging information between Windows applications. Below is an example of reading data from EXCEL into SAS.

```
/******  
/* NAME: DDE2 *  
/* TITLE: USING DDE TO READ DATA FROM EXCEL *  
/* DESC: DEMONSTRATES HOW TO USE DDE TO READ DATA FROM EXCEL *  
/* INTO SAS. TO EXECUTE THIS PROGRAM YOU MUST *  
/* FIRST INVOKE MICROSOFT EXCEL AND OPEN UP A NEW *  
/* WORKSHEET CALLED "SHEET1.XLS. ENTER NUMERIC DATA IN *  
/* EXCEL WORKSHEET IN ROWS 1-3 , COLUMNS 1-3. *  
/******  
  
/* THE DDE LINK IS ESTABLISHED USING MICROSOFT EXCEL SHEET1,  
   ROWS 1 THROUGH 3 AND COLUMNS 1 THROUGH 3. */
```

```
FILENAME MONTHLY DDE 'EXCEL | SHEET1.XLS ! R1C1:R3C3';  
DATA MONTHLY;  
  INFILE MONTHLY;  
  INPUT VAR1 VAR2 VAR3;  
RUN;  
PROC PRINT;  
RUN;
```

Below is an example of sending data from SAS to EXCEL .

```
/******  
/* NAME: DDE1 *  
/* TITLE: USING DDE TO SEND DATA TO EXCEL *  
/* DESC: DEMONSTRATES HOW USE DDE TO SEND DATA FROM SAS TO *  
/* EXCEL. TO EXECUTE THIS PROGRAM YOU MUST FIRST *  
/* INVOKE MICROSOFT EXCEL AND OPEN UP A NEW WORKSHEET *  
/* CALLED "SHEET1" . *  
/******  
  
/* THE DDE LINK IS ESTABLISHED USING MICROSOFT EXCEL SHEET1,  
   ROWS 1 THROUGH 100 AND COLUMNS 1 THROUGH 3. */
```

```
FILENAME RANDOM DDE 'EXCEL | SHEET1 ! R1C1:R100C3';  
  
DATA RANDOM;  
  FILE RANDOM;  
  DO I=1 TO 100;  
    X=RANUNI(I);  
    Y=10+X;  
    Z=X-10;  
    PUT X Y Z;  
  END;  
RUN;
```