Chapter 16

Reading Raw Data in Fixed Fields

Overview

Raw data can be organized in several ways. They can be arranged in **columns** or **fixed fields** (i.e., the data are arranged in columns and the values for a particular field begin and end in the same columns) or in a **free-format**.

Fixed-field data example

```
1---+----10---+----20---+----
BIRD FEEDER    LG088   3 $29.95
GLASS MUGS     SB082   6 $25.00
GLASS TRAY     BQ049  12 $39.95
PADDED HANGRS  MN256  15 $9.95
JEWELRY BOX    AJ498  23 $45.00
RED APRON      AQ072   9 $6.50
CRYSTAL VASE   AQ672  27 $29.95
PICNIC BASKET  LS930  21 $15.00
```

Free-format data example

```
1---+----10---+----20---+--
BARNES NORTHWEST 36098.45
FARLSON SOUTHWEST 24394.09
LAWRENCE NORTHEAST 19504.26
NELSON SOUTHEAST 16930.84
STEWART MIDWEST 23845.13
TAYLOR MIDWEST 12354.42
TREADWAY SOUTHWEST 41092.84
WALSTON SOUTHEAST 28938.71
```

How your data is organized and what type of data you have determine which **input style** you should use to read the data.

Raw Data Input Styles

SAS provides three primary input styles:

- Column input
- Formatted input
- List input

This chapter deals with how to use column input and formatted input to read standard and nonstandard data that is arranged in fixed fields.

Review of Column Input

You can use column input to read the values for Item, IDnum, InStock, and BackOrd from the raw data file Invent.

```
input Item $ 1-13 IDnum $ 15-19 InStock 21-22 BackOrd 24-25;
```

Raw Data File Invent

```
1---+----10---+----20---+--
BIRD FEEDER    LG088  3 20
GLASS MUGS     SB082  6 12
GLASS TRAY     BQ049 12  6
PADDED HANGRS  MN256 15 20
JEWELRY BOX    AJ498 23  0
RED APRON      AQ072  9 12
CRYSTAL VASE   AQ672 27  0
PICNIC BASKET  LS930 21  0
```

Features of Column Input

- It can read fields in any order.
- It can be used to read character variable values that contain embedded blanks.
- No placeholder is required for missing data.
- Fields or parts of fields can be re-read.
- Fields do not have to be separated by blanks or other delimiters.

Read Fields in Any Order

```
input InStock 21-22 BackOrd 24-25 Item $ 1-13 IDnum $ 15-19;
```

Raw Data File Invent

```
1---+----10---+----20---+--
BIRD FEEDER    LG088  3 20
GLASS MUGS     SB082  6 12
GLASS TRAY     BQ049 12  6
PADDED HANGRS  MN256 15 20
JEWELRY BOX    AJ498 23  0
RED APRON      AQ072  9 12
CRYSTAL VASE   AQ672 27  0
PICNIC BASKET  LS930 21  0
```

InStock	BackOrd	Item	IDnum
3	20	BIRD FEEDER	LG088
6	12	GLASS MUGS	SB082
12	6	GLASS TRAY	BQ049
15	20	PADDED HANGRS	MN256
23	0	JEWELRY BOX	AJ498
9	12	RED APRON	AQ072
27	0	CRYSTAL VASE	AQ672
21	0	PICNIC BASKET	LS930

Read Character Variable Values That Contain Embedded Blanks

input Name $ 1-25;

```
1---+----10---+----20---+-
JOSEPH PAUL THACKERY JR.
```

No Placeholder Is Required for Missing Data

input Item $ 1-13 IDnum $ 15-19
InStock 21-22 BackOrd 24-25;

```
1---+----10---+----20---+--
PADDED HANGRS  MN256 15  20
JEWELRY BOX    AJ498      0
RED APRON      AQ072  9  12
```

A blank field is read as missing and does not cause other fields to be read incorrectly.

Fields or Parts of Fields Can Be Re-read

input Item $ 1-13 **IDnum $ 15-19** **Supplier $ 15-16**
InStock 21-22 BackOrd 24-25;

```
1---+----10---+----20---+--
PADDED HANGRS  MN256 15  20
JEWELRY BOX    AJ498 23   0
RED APRON      AQ072  9  12
```

No Blanks or Other Delimiters Required to Separate the Fields

```
input Item $ 1-13 IDnum $ 14-18 InStock 19-20
      BackOrd 21-22;
```

```
1---+----10---+----20---+--
PADDED HANGRSMN2561520
JEWELRY BOX   AJ49823 0
RED APRON     AQ072 912
```

Review: Standard *vs.* Nonstandard Numeric Data

Standard numeric data

- Numbers
- Decimal points
- Numbers in scientific, or E, notation (23E4)
- Minus signs and plus signs.

Nonstandard numeric data

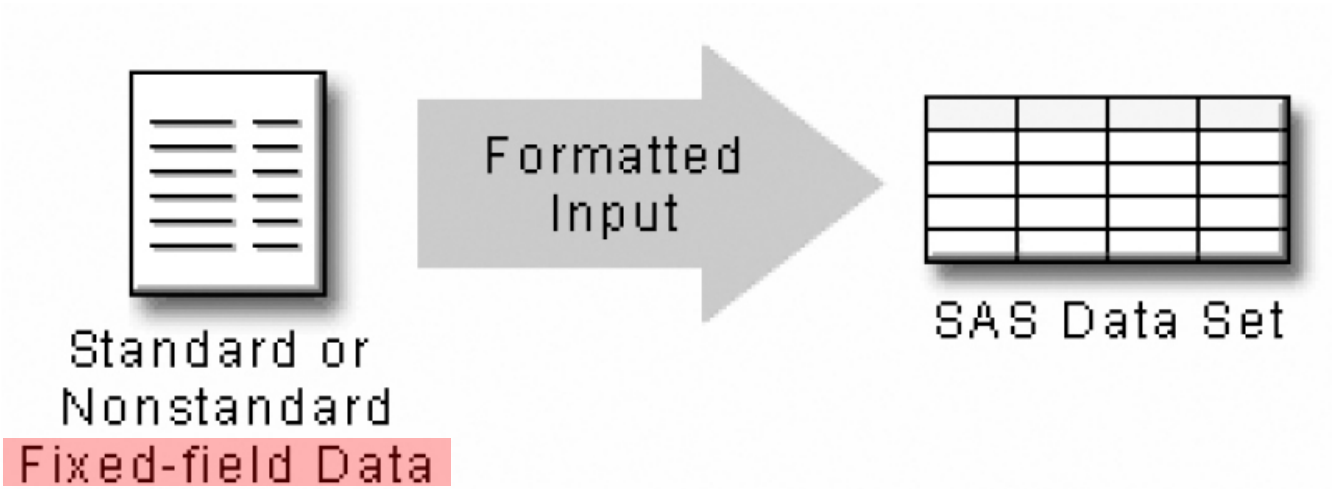
- Values that contain special characters, such as percent signs (%), dollar signs ($), and commas (,)
- Date and time values
- Data in fraction, integer binary, real binary, and hexadecimal forms.

Raw Data File Empdata

```
1---+----10---+----20---+---
EVANS    DONNY 112 29,996.63
HELMS    LISA  105 18,567.23
HIGGINS  JOHN  111 25,309.00
LARSON   AMY   113 32,696.78
MOORE    MARY  112 28,945.89
POWELL   JASON 103 35,099.50
RILEY    JUDY  111 25,309.00
```

Choosing an Input Style

Nonstandard data values require an input style that has more flexibility than column input. You can use **formatted input**, which combines the features of column input with the ability to read both standard and nonstandard fixed-field data.



Formatted Input

General form:

INPUT *<pointer-control> variable informat.;*

where

- pointer-control positions the input pointer on a specified column. There are two column pointer controls:

  1. ***@n*** moves the input pointer to a specific column number, n
  2. ***+n*** moves the input pointer forward n columns from the current position

- informat. a special instruction that specifies how SAS reads the raw data.

Using the @*n* Column Pointer Control

input LastName $7. **@9** FirstName $5.;

```
1---+---v10---+----20---+---
EVANS    DONNY  112 29,996.63
HELMS    LISA   105 18,567.23
HIGGINS  JOHN   111 25,309.00
LARSON   AMY    113 32,696.78
MOORE    MARY   112 28,945.89
POWELL   JASON  103 35,099.50
RILEY    JUDY   111 25,309.00
```

Why is there no pointer-control for the variable LastName?

Reading Columns in Any Order With *@n*

With @n, you can read columns in any order by moving a pointer forward or backward when reading a record.

```
input @9 FirstName $5. @1 LastName $7. @15 JobTitle 3. @19
      Salary comma9.;
```

```
1---+---v10---+----20---+---
EVANS    DONNY 112 29,996.63
HELMS    LISA  105 18,567.23
HIGGINS  JOHN  111 25,309.00
```

```
v---+----10---+----20---+---
EVANS    DONNY 112 29,996.63
HELMS    LISA  105 18,567.23
HIGGINS  JOHN  111 25,309.00
```

```
1---+----10---v---v20---+---
EVANS    DONNY 112 29,996.63
HELMS    LISA  105 18,567.23
HIGGINS  JOHN  111 25,309.00
```

Using the +n Pointer Control

```
input LastName $7. +1 FirstName $5. +5 Salary comma9.
      @15 JobTitle 3.;
```

```
v---+----10---+----20---+---
EVANS    DONNY 112 29,996.63
HELMS    LISA  105 18,567.23
HIGGINS  JOHN  111 25,309.00
```

```
1---+--v-10---+----20---+---
EVANS    DONNY 112 29,996.63
HELMS    LISA  105 18,567.23
HIGGINS  JOHN  111 25,309.00
```

```
1---+---v10---+----20---+---
EVANS    DONNY 112 29,996.63
HELMS    LISA  105 18,567.23
HIGGINS  JOHN  111 25,309.00
```

```
                    12345
1---+----10---v---v20---+---
EVANS    DONNY 112 29,996.63
HELMS    LISA  105 18,567.23
HIGGINS  JOHN  111 25,309.00
```

```
1---+----10---v----20---+---
EVANS    DONNY 112 29,996.63
HELMS    LISA  105 18,567.23
HIGGINS  JOHN  111 25,309.00
```

Note:

- know where the column pointer is located after each data value is read.
- The **+(-*n*)** pointer control can be used to move the pointer backwards n columns.

Some Informats

PERCENTw.d	DATEw.	NENGOw.
$BINARYw.	DATETIMEw.	PDw.d
$VARYINGw.	HEXw.	PERCENTw.
$w.	JULIANw.	TIMEw.
COMMAw.d	MMDDYYw.	w.d

Note that

- each informat contains a *w* value to indicate the width of the raw data field.
- each informat also contains a period, which is a required delimiter.
- for some informats, the optional *d* value specifies the number of implied decimal places.
- informats for reading character data always begin with a dollar sign ($).

Reading Character Values With Informats

You use the $w. informat to read character data, where *w* is the field width of the data value (total # of columns).

```
input @9 FirstName $5.;
```

```
                  12345
1---+---V10---+----20---+---
EVANS     DONNY  112 29,996.63
HELMS     LISA   105 18,567.23
HIGGINS   JOHN   111 25,309.00
LARSON    AMY    113 32,696.78
MOORE     MARY   112 28,945.89
POWELL    JASON  103 35,099.50
RILEY     JUDY   111 25,309.00
RYAN      NEAL   112 28,180.00
WILSON    HENRY  113 31,875.46
WOODS     CHIP   105 17,098.71
```

Reading Standard Numeric Values With Informats

You use the **w.d** informat to read standard numeric data , where *w* is the field width of the data value (total # of columns), and *d* optionally specifies the number of implied decimal places for the value. Note: The *w.d* informat ignores any specified *d* value if the data already contains a decimal point.

Raw Data Value	w. Informat	Variable Value
34.0008	7.	34.0008

input @9 FirstName $5. @1 LastName $7. +7 JobTitle **3.**;

```
                                  123
1---+----10---+----20---+---
EVANS    DONNY  112  29,996.63
HELMS    LISA   105  18,567.23
HIGGINS  JOHN   111  25,309.00
LARSON   AMY    113  32,696.78
MOORE    MARY   112  28,945.89
POWELL   JASON  103  35,099.50
RILEY    JUDY   111  25,309.00
```

Reading Nonstandard Numeric Values With Informats

You use the **COMMAw.d** informat to read nonstandard numeric values and to remove embedded blanks, commas, dashes, dollar signs, right parentheses, left parentheses, etc. This informat has three parts:

1.	The informat name	COMMA
2.	A value that specifies the width of the field to be read (including dollar signs, decimal places, or other special characters), followed by a period	w.
3.	An optional value that specifies the number of implied decimal places for a value (not effective if the value already contains decimal places).	d

Reading Nonstandard Numeric Values With Informats

Example:

```
data sasuser.empinfo;
    infile empdata;
    input @9 FirstName $5. @1 LastName $7. +7 JobTitle 3.
          @19 Salary comma9.;
run;
proc print data=sasuser.empinfo;
run;
```

```
                                    123456789
1---+----10---+---v20---+-v-
EVANS    DONNY  112  29,996.63
HELMS    LISA   105  18,567.23
HIGGINS  JOHN   111  25,309.00
LARSON   AMY    113  32,696.78
MOORE    MARY   112  28,945.89
POWELL   JASON  103  35,099.50
RILEY    JUDY   111  25,309.00
```

Obs	FirstName	LastName	JobTitle	Salary
1	DONNY	EVANS	112	29996.63
2	ALISA	HELMS	105	18567.23
3	JOHN	HIGGINS	111	25309.00
4	AMY	LARSON	113	32696.78
5	MARY	MOORE	112	28945.89
6	JASON	POWELL	103	35099.50
7	JUDY	RILEY	111	25309.00

DATA Step Processing of Informats

During the compile phase, the **character variables** are defined with the exact length specified by the informat, but the lengths for **numeric variables**, JobTitle and Salary below, in the program data vector are different from the lengths that are specified by their informats (the # of columns to read from the raw date). By default, SAS stores **numeric values** (no matter how many digits the value contains) as floating-point numbers in **8 bytes** of storage. The length of a stored numeric variable is not affected by an informat's width.

```
data sasuser.empinfo;
    infile empdata;
    input @9 FirstName $5. @1 LastName $7. +7 JobTitle 3.
          @19 Salary comma9.;
run;
```

```
                                123 123456789
1---+----10---V-V-V20---+-V-
EVANS    DONNY 112 29,996.63
HELMS    LISA  105 18,567.23
HIGGINS  JOHN  111 25,309.00
```

Program Data Vector

_N_	_ERROR_	FirstName	LastName	JobTitle	Salary
		$5.	$7.	8.	8.
		DONNY	EVANS	112	29996.63

DATA Step Processing of Informats

What happens if you change the widths of the two numeric variables to 8?

```
data sasuser.empinfo;
    infile empdata;
    input @9 FirstName $5. @1 LastName $7. +7 JobTitle 8.
          @19 Salary comma8.;
run;
```

```
                    12345678
                        12345678
1---+----10---V---V20V--+V--
EVANS    DONNY  112 29,996.63
HELMS    LISA   105 18,567.23
HIGGINS  JOHN   111 25,309.00
```

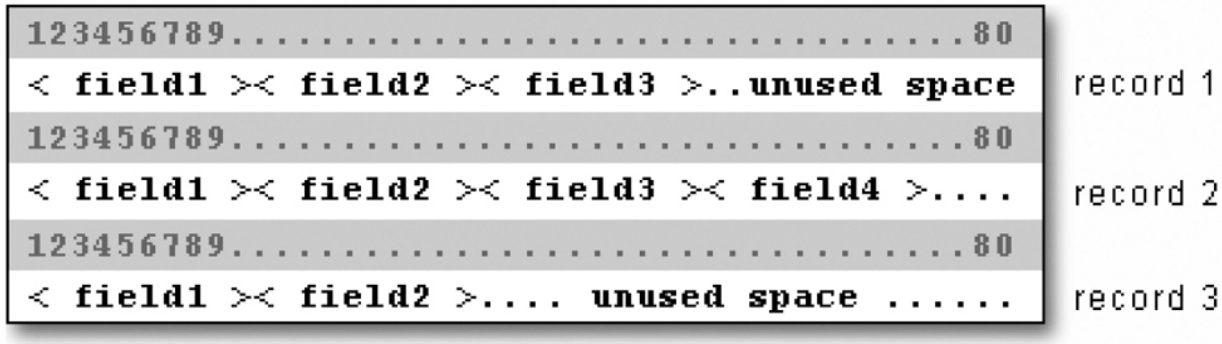
Program Data Vector

_N_	_ERROR_	FirstName	LastName	JobTitle	Salary
		$5.	$7.	8.	8.
		DONNY	EVANS	•	29996.6

Record Formats

A record format specifies how records are organized in a file. This format might affect how data is read with column input and formatted input. Two common record formats are fixed-length records and variable-length records.

- **Fixed-Length Records :** External files have an end-of-record marker after a predetermined number of columns. A typical record length is 80 columns.

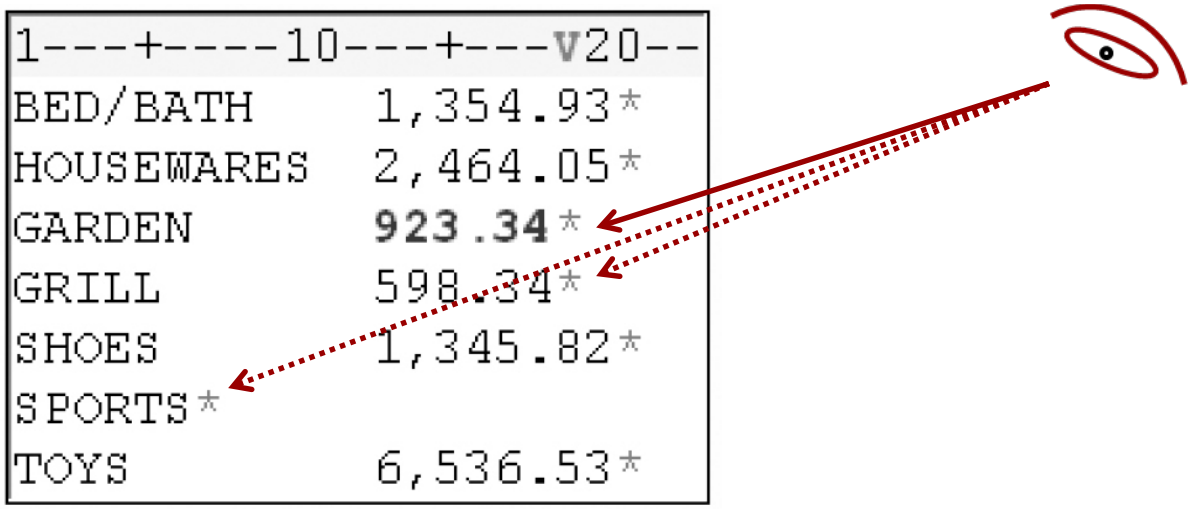


- **Variable-Length Records :** External files have an end-of-record marker after the last field in each record.

Reading Variable-Length Records

When you work with variable-length records that contain fixed-field data, you might have values that are **shorter** than others or that are **missing**. This can cause problems when you try to read the raw data into your SAS data set.

input Dept $ 1-11 @13 Receipts comma8.;



The INPUT statement specifies a field width of 8 columns for Receipts. In the third record, the input pointer encounters an end-of-record marker before the 8th column. The input pointer moves down to the next record in an attempt to complete the value for Receipts. However, **GRILL** is a character value, and Receipts is a numeric variable. Thus, an invalid data error occurs, and Receipts is set to missing.

Use the PAD Option

You use the **PAD** option in the INFILE statement to avoid the problem. It pads each record with blanks so that all data lines have the same length to the length of the **LRECL=** option (default = 80 bytes).

```
infile empdata pad;
input Dept $ 1-11 @13 Receipts comma8.;
```

```
1---+----10---+----20--
BED/BATH     1,354.93
HOUSEWARES   2,464.05
GARDEN         923.34
GRILL          598.34
SHOES        1,345.82
SPORTS
TOYS         6,536.53
```

Note:

- The PAD option is useful only when missing data occurs at the end of a record or when SAS encounters an end-of-record marker before all fields are completely read.
- The default value of the maximum record length is determined by your operating system. If you encounter unexpected results when reading many variables, you might need to change the maximum record length by specifying the LRECL=option in the INFILE statement.