Title

stata.com

infile (free format) — Read unformatted text data

Description

`infile` reads into memory from a disk a dataset that is not in Stata format.

Here we discuss using `infile` to read free-format data, meaning datasets in which Stata does not need to know the formatting information. Another variation on `infile` allows reading fixed-format data; see [D] **infile (fixed format)**. Yet another alternative is `import delimited`, which is easier to use if your data are tab- or comma-separated and contain 1 observation per line. Stata has other commands for reading data, too. If you are not certain that `infile` will do what you are looking for, see [D] **import** and [U] **21 Entering and importing data**.

After the data are read into Stata, they can be saved in a Stata-format dataset; see [D] **save**.

Quick start

Import unformatted text data from `mydata1.raw` and name the imported `float` variables `v1`, `v2`, and `v3`

```
infile v1 v2 v3 using mydata1
```

As above, but skip 1 variable in the original file between `v1` and `v2`

```
infile v1 _skip(1) v2 v3 using mydata1
```

As above, and indicate that `v1` is a `byte` variable, `v2` is a string variable of length 30, and `v3` is a `double` variable

```
infile byte v1 _skip(1) str30 v2 double v3 using mydata1
```

Also read `v4` as a `double`

```
infile byte v1 _skip(1) str30 v2 double(v3 v4) using mydata1
```

Import unformatted text data from `mydata2.raw` where 74 observations on `v1`, `v2`, and `v3` are stored in rows instead of columns

```
infile v1 v2 v3 using mydata2, byvariable(74)
```

As above, but import `mydata2.csv`

```
infile v1 v2 v3 using mydata2.csv, byvariable(74)
```

Menu

File > Import > Unformatted text data

Syntax

```
infile varlist [ _skip[(#)] ] [ varlist [ _skip[(#)] ... ] ] using filename [ if ] [ in ]
    [ , options ]
```

If *filename* is specified without an extension, `.raw` is assumed. If *filename* contains embedded spaces, remember to enclose it in double quotes.

options	Description
Main	
`clear`	replace data in memory
Options	
`automatic`	create value labels from nonnumeric data
`byvariable(#)`	organize external file by variables; # is number of observations

Options

Main

`clear` specifies that it is okay for the new data to replace the data that are currently in memory. To ensure that you do not lose something important, `infile` will refuse to read new data if data are already in memory. `clear` allows `infile` to replace the data in memory. You can also drop the data yourself by typing `drop _all` before reading new data.

Options

`automatic` causes Stata to create value labels from the nonnumeric data it reads. It also automatically widens the display format to fit the longest label.

`byvariable(#)` specifies that the external data file is organized by variables rather than by observations. All the observations on the first variable appear, followed by all the observations on the second variable, and so on. Time-series datasets sometimes come in this format.

Remarks and examples

stata.com

This section describes `infile` features for reading data in free or comma-separated–value format.

Remarks are presented under the following headings:

Reading free-format data

In free format, data are separated by one or more white-space characters—blanks, tabs, or new lines (carriage return, line feed, or carriage-return/line feed combinations). Thus one observation may span any number of lines.

Numeric missing values are indicated by single periods (“.”).

▷ Example 1

In the file `highway.raw`, we have information on the accident rate per million vehicle miles along a stretch of highway, the speed limit on that highway, and the number of access points (on-ramps and off-ramps) per mile. Our file contains

begin highway.raw, example 1

```
4.58 55 4.6
2.86  60 4.4
1.61 . 2.2
3.02 60
4.7
```

end highway.raw, example 1

We can read these data by typing

```
. infile acc_rate spdlimit acc_pts using highway
(4 observations read)
. list
```

	acc_rate	spdlimit	acc_pts
1.	4.58	55	4.6
2.	2.86	60	4.4
3.	1.61	.	2.2
4.	3.02	60	4.7

The spacing of the numbers in the original file is irrelevant.

◁

❑ Technical note

Missing values need not be indicated by one period. The third observation on the speed limit is missing in example 1. The raw data file indicates this by recording one period. Let’s assume, instead, that the missing value was indicated by the word `unknown`. Thus the raw data file appears as

begin highway.raw, example 2

```
4.58 55 4.6
2.86  60 4.4
1.61 unknown 2.2
3.02 60
4.7
```

end highway.raw, example 2

Here is the result of infiling these data:

```
. infile acc_rate spdlimit acc_pts using highway
'unknown' cannot be read as a number for spdlimit[3]
(4 observations read)
```

`infile` warned us that it could not read the word `unknown`, stored a *missing*, and then continued to read the rest of the dataset. Thus aside from the warning message, results are unchanged.

Because not all packages indicate missing data in the same way, this feature can be useful when reading data. Whenever `infile` sees something that it does not understand, it warns you, records a *missing*, and continues. If, on the other hand, the missing values were recorded not as `unknown` but as, say, `99`, Stata would have had no difficulty reading the number, but it would also have stored 99 rather than missing. To convert such coded missing values to true missing values, see [D] **mvencode**.

❑

Reading comma-separated data

In comma-separated–value format, data are separated by commas. You may mix comma-separated–value and free formats. Missing values are indicated either by single periods or by multiple commas that serve as placeholders, or both. As with free format, 1 observation may span any number of input lines.

▷ Example 2

We can modify the format of `highway.raw` used in example 1 without affecting `infile`'s ability to read it. The dataset can be read with the same command, and the results would be the same if the file instead contained

begin highway.raw, example 3

```
4.58,55 4.6
2.86, 60,4.4
1.61,,2.2
3.02,60
4.7
```

end highway.raw, example 3

◁

Specifying variable types

The variable names you type after the word `infile` are new variables. The syntax for a new variable is

$$\left[\,\textit{type}\,\right]\ \textit{new\_varname}\left[\,\underline{:}\textit{label\_name}\,\right]$$

A full discussion of this syntax can be found in [U] **11.4 varlists**. As a quick review, new variables are, by default, of type `float`. This default can be overridden by preceding the variable name with a storage type (`byte`, `int`, `long`, `float`, `double`, or `str#`) or by using the `set type` command. A list of variables placed in parentheses will be given the same type. For example,

`double`(*first_var second_var ... last_var*)

causes *first_var second_var ... last_var* to all be of type `double`.

There is also a shorthand syntax for variable names with numeric suffixes. The varlist `var1-var4` is equivalent to specifying `var1 var2 var3 var4`.

▷ Example 3

In the highway example, we could `infile` the data `acc_rate`, `spdlimit`, and `acc_pts` and force the variable `spdlimit` to be of type `int` by typing

```
. infile acc_rate int spdlimit acc_pts using highway, clear
(4 observations read)
```

We could force all variables to be of type `double` by typing

```
. infile double(acc_rate spdlimit acc_pts) using highway, clear
(4 observations read)
```

We could call the three variables `v1`, `v2`, and `v3` and make them all of type `double` by typing

```
. infile double(v1-v3) using highway, clear
(4 observations read)
```

◁

Reading string variables

By explicitly specifying the types, you can read string variables, as well as numeric variables.

▷ Example 4

Typing `infile str20 name age sex using myfile` would read

begin myfile.raw

```
"Sherri Holliday" 25 1
Branton 32 1
"Bill Ross" 27,0
```

begin myfile.raw

or even

begin myfile.raw, variation 2

```
'Sherri Holliday' 25,1 "Branton" 32
1,'Bill Ross', 27,0
```

end myfile.raw, variation 2

The spacing is irrelevant, and either single or double quotes may be used to delimit strings. The quotes do not count when calculating the length of strings. Quotes may be omitted altogether if the string contains no blanks or other special characters (anything other than letters, numbers, or underscores).

Typing

```
. infile str20 name age sex using myfile, clear
(3 observations read)
```

makes `name` a `str20` and `age` and `sex` `float`s. We might have typed

```
. infile str20 name age int sex using myfile, clear
(3 observations read)
```

to make `sex` an `int` or

```
. infile str20 name int(age sex) using myfile, clear
(3 observations read)
```

to make both `age` and `sex` `int`s.

◁

❑ Technical note

`infile` can also handle nonnumeric data by using *value labels*. We will briefly review value labels, but you should see **[U] 12.6.3 Value labels** for a complete description.

A value label is a mapping from the set of integers to words. For instance, if we had a variable called `sex` in our data that represented the sex of the individual, we might code 0 for male and 1 for female. We could then just remember that every time we see a value of 0 for `sex`, that observation refers to a male, whereas 1 refers to a female.

Even better, we could inform Stata that 0 represents males and 1 represents females by typing

```
. label define sexfmt 0 "Male" 1 "Female"
```

Then we must tell Stata that this coding scheme is to be associated with the variable `sex`. This is typically done by typing

```
. label values sex sexfmt
```

Thereafter, Stata will print `Male` rather than 0 and `Female` rather than 1 for this variable.

Stata has the ability to turn a value label around. Not only can it go from numeric codes to words such as "Male" and "Female", it can also go from the words to the numeric code. We tell `infile` the value label that goes with each variable by placing a colon (`:`) after the variable name and typing the name of the value label. Before we do that, we use the `label define` command to inform Stata of the coding.

Let's assume that we wish to `infile` a dataset containing the words `Male` and `Female` and that we wish to store numeric codes rather than the strings themselves. This will result in considerable data compression, especially if we store the numeric code as a `byte`. We have a dataset named `persons.raw` that contains name, sex, and age:

begin persons.raw

```
"Arthur Doyle" Male 22
"Mary Hope" Female 37
"Guy Fawkes" Male 48
"Carrie House" Female 25
```

end persons.raw

Here is how we read and encode it at the same time:

```
. label define sexfmt 0 "Male" 1 "Female"
. infile str16 name sex:sexfmt age using persons
(4 observations read)
. list
```

	name	sex	age
1.	Arthur Doyle	Male	22
2.	Mary Hope	Female	37
3.	Guy Fawkes	Male	48
4.	Carrie House	Female	25

The str16 in the infile command applies only to the name variable; sex is a numeric variable, which we can prove by typing

```
. list, nolabel
```

	name	sex	age
1.	Arthur Doyle	0	22
2.	Mary Hope	1	37
3.	Guy Fawkes	0	48
4.	Carrie House	1	25

❑

❑ Technical note

When infile is directed to use a value label and it finds an entry in the file that does not match any of the codings recorded in the label, it prints a warning message and stores *missing* for the observation. By specifying the automatic option, you can instead have infile automatically add new entries to the value label.

Say that we have a dataset containing three variables. The first, region of the country, is a character string; the remaining two variables, which we will just call var1 and var2, contain numbers. We have stored the data in a file called geog.raw:

begin geog.raw

```
"NE"       31.23      87.78
'NCntrl'   29.52      98.92
South      29.62     114.69
West       28.28     218.92
NE         17.50      44.33
NCntrl     22.51      55.21
```

end geog.raw

The easiest way to read this dataset is to type

```
. infile str6 region var1 var2 using geog
```

making region a string variable. We do not want to do this, however, because we are practicing for reading a dataset like this containing 20,000 observations. If region were numerically encoded and stored as a byte, there would be a 5-byte saving per observation, reducing the size of the data by 100,000 bytes. We also do not want to bother with first creating the value label. Using the automatic option, infile creates the value label automatically as it encounters new regions.

```
. infile byte region:regfmt var1 var2 using geog, automatic clear
(6 observations read)
. list, sep(0)
```

	region	var1	var2
1.	NE	31.23	87.78
2.	NCntrl	29.52	98.92
3.	South	29.62	114.69
4.	West	28.28	218.92
5.	NE	17.5	44.33
6.	NCntrl	22.51	55.21

infile automatically created and defined a new value label called regfmt. We can use the label list command to view its contents:

```
. label list regfmt
regfmt:
                1 NE
                2 NCntrl
                3 South
                4 West
```

The value label need not be undefined before we use infile with the automatic option. If the value label regfmt had been previously defined as

```
. label define regfmt 2 "West"
```

the result of label list after the infile would have been

```
regfmt:
                2 West
                3 NE
                4 NCntrl
                5 South
```

The automatic option is convenient, but there is one reason for using it. Suppose that we had a dataset containing, among other things, information about an individual's sex. We know that the sex variable is supposed to be coded male and female. If we read the data by using the automatic option and if one of the records contains fmlae, then infile will blindly create a third sex rather than print a warning.

❑

Skipping variables

Specifying _skip instead of a variable name directs infile to ignore the variable in that location. This feature makes it possible to extract manageable subsets from large disk datasets. A number of contiguous variables can be skipped by specifying _skip(#), where # is the number of variables to ignore.

▷ Example 5

In the highway example from example 1, the data file contained three variables: acc_rate, spdlimit, and acc_pts. We can read the first two variables by typing

```
. infile acc_rate spdlimit _skip using highway
(4 observations read)
```

We can read the first and last variables by typing

```
. infile acc_rate _skip acc_pts using highway, clear
(4 observations read)
```

We can read the first variable by typing

```
. infile acc_rate _skip(2) using highway, clear
(4 observations read)
```

_skip may be specified more than once. If we had a dataset containing four variables—say, a, b, c, and d—and we wanted to read just a and c, we could type infile a _skip c _skip using *filename*.

◁

Skipping observations

Subsets of observations can be extracted by specifying `if` *exp*, which also makes it possible to extract manageable subsets from large disk datasets. Do not, however, use the *_variable* `_N` in *exp*. Use the `in` *range* qualifier to refer to observation numbers within the disk dataset.

▷ Example 6

Again referring to the highway example, if we type

```
. infile acc_rate spdlimit acc_pts if acc_rate>3 using highway, clear
(2 observations read)
```

only observations for which `acc_rate` is greater than 3 will be infiled. We can type

```
. infile acc_rate spdlimit acc_pts in 2/4 using highway, clear
(eof not at end of obs)
(3 observations read)
```

to read only the second, third, and fourth observations.

◁

Reading time-series data

If you are dealing with time-series data, you may receive datasets organized by variables rather than by observations. All the observations on the first variable appear, followed by all the observations on the second variable, and so on. The `byvariable(#)` option specifies that the external data file is organized in this way. You specify the number of observations in the parentheses, because `infile` needs to know that number to read the data properly. You can also mark the end of one variable's data and the beginning of another's data by placing a semicolon ("`;`") in the raw data file. You may then specify a number larger than the number of observations in the dataset and leave it to `infile` to determine the actual number of observations. This method can also be used to read unbalanced data.

▷ Example 7

We have time-series data on 4 years recorded in the file `time.raw`. The dataset contains information on year, amount, and cost, and is organized by variable:

begin time.raw

```
1980 1981 1982 1983
14 17 25 30
120 135 150
180
```

end time.raw

We can read these data by typing

```
. infile year amount cost using time, byvariable(4) clear
(4 observations read)
. list
```

	year	amount	cost
1.	1980	14	120
2.	1981	17	135
3.	1982	25	150
4.	1983	30	180

If the data instead contained semicolons marking the end of each series and had no information for amount in 1983, the raw data might appear as

```
1980 1981 1982 1983 ;
14 17 25 ;
120 135 150
180 ;
```

We could read these data by typing

```
. infile year amount cost using time, byvariable(100) clear
(4 observations read)
. list
```

	year	amount	cost
1.	1980	14	120
2.	1981	17	135
3.	1982	25	150
4.	1983	.	180

◁

Also see

[D] **infile (fixed format)** — Read text data in fixed format with a dictionary

[D] **export** — Overview of exporting data from Stata

[D] **import** — Overview of importing data into Stata

[U] **21 Entering and importing data**