Title

stata.com

import delimited — Import and export delimited text data

Description

`import delimited` reads into memory a text file in which there is one observation per line and the values are separated by commas, tabs, or some other delimiter. The two most common types of text data to import are comma-separated values (`.csv`) text files and tab-separated text files, often `.txt` files. Similarly, `export delimited` writes Stata's data to a text file.

Stata has other commands for importing data. If you are not sure that `import delimited` will do what you are looking for, see [D] **import** and **[U] 22 Entering and importing data**.

Quick start

Load comma-delimited `mydata.csv` with 2 variables to be named `v1` and `v2`

```
import delimited v1 v2 using mydata
```

As above, but with variable names on the first row

```
import delimited mydata
```

As above, but with variable names in row 5 and an ignorable header in the first 4 rows

```
import delimited mydata, varnames(5)
```

Load only columns 2 to 300 and the first 1,000 rows with variable names in row 1

```
import delimited mydata, colrange(2:300) rowrange(:1000)
```

Load tab-delimited data from `mydata.txt`

```
import delimited mydata.txt, delimiters(tab)
```

Load semicolon-delimited data from `mydata.txt`

```
import delimited mydata.txt, delimiters(";")
```

Force columns 2 to 6 to be read as string to preserve leading zeros

```
import delimited mydata, stringcols(2/6)
```

Export data in memory to `mydata.csv`

```
export delimited mydata
```

As above, but export only `v1` and `v2`

```
export delimited v1 v2 using mydata
```

As above, but output numeric values for variables with value labels

```
export delimited v1 v2 using mydata, nolabel
```

Menu

import delimited

File > Import > Text data (delimited, *.csv, ...)

export delimited

File > Export > Text data (delimited, *.csv, ...)

Syntax

Load a delimited text file

```
import delimited [using] filename [, import_delimited_options]
```

Rename specified variables from a delimited text file

```
import delimited extvarlist using filename [, import_delimited_options]
```

Save data in memory to a delimited text file

```
export delimited [using] filename [if] [in] [, export_delimited_options]
```

Save subset of variables in memory to a delimited text file

```
export delimited [varlist] using filename [if] [in] [, export_delimited_options]
```

If *filename* is specified without an extension, `.csv` is assumed for both `import delimited` and `export delimited`. If *filename* contains embedded spaces, enclose it in double quotes.

extvarlist specifies variable names of imported columns.

import_delimited_options	Description
rowrange(\[*start* \]\[:*end* \])	row range of data to load
colrange(\[*start* \]\[:*end* \])	column range of data to load
varnames(# | nonames)	treat row # of data as variable names or the data do not have variable names
case(preserve | lower | upper)	preserve the case or read variable names as lowercase (the default) or uppercase
asfloat	import all floating-point data as floats
asdouble	import all floating-point data as doubles
encoding(*encoding*)	specify the encoding of the text file being imported
bindquotes(loose | strict | nobind)	specify how to handle double quotes in data
stripquotes(yes | no | default)	remove or keep double quotes in data
delimiters("*chars*"\[, collapse | asstring \])	use *chars* as delimiters
parselocale(*locale*)	specify the locale to use for interpreting numbers in the text file being imported
decimalseparator(*character*)	character to use for the decimal separator when parsing numbers
groupseparator(*character*)	character to use for the grouping separator when parsing numbers
maxquotedrows(# | unlimited)	number of rows of data allowed inside a quoted string when bindquote(strict) is specified
numericcols(*numlist* | _all)	force specified columns to be numeric
stringcols(*numlist* | _all)	force specified columns to be string
clear	replace data in memory
favorstrfixed	favor storing string variables as str# rather than strL

favorstrfixed does not appear in the dialog box.

export_delimited_options	Description
Main	
delimiter("*char*" | tab)	use *char* as delimiter
novarnames	do not write variable names on the first line
nolabel	output numeric values (not labels) of labeled variables
datafmt	use the variables' display format upon export
quote	always enclose strings in double quotes
replace	overwrite existing *filename*

Options for import delimited

rowrange(\[*start* \]\[:*end* \]) specifies a range of rows within the data to load. *start* and *end* are integer row numbers.

colrange(\[*start* \]\[:*end* \]) specifies a range of variables within the data to load. *start* and *end* are integer column numbers.

`varnames(#|nonames)` specifies where or whether variable names are in the data. By default, `import delimited` tries to determine whether the file includes variable names. `import delimited` translates the names in the file to valid Stata variable names. The original names from the file are stored unmodified as variable labels.

`varnames(#)` specifies that the variable names are in row # of the data; any data before row # should not be imported.

`varnames(nonames)` specifies that the variable names are not in the data.

`case(preserve|lower|upper)` specifies the case of the variable names after import. The default is `case(lowercase)`.

`asfloat` imports floating-point data as type `float`. The default storage type of the imported variables is determined by `set type`.

`asdouble` imports floating-point data as type `double`. The default storage type of the imported variables is determined by `set type`.

`encoding(`*encoding*`)` specifies the encoding of the text file to be read. The default is `encoding("latin1")`. Specify `encoding("utf-8")` for the files to be encoded in UTF-8. `import delimited` uses Java encoding. A list of available encodings can be found at https://docs.oracle.com/javase/8/docs/technotes/guides/intl/encoding.doc.html.

Option `charset()` is a synonym for `encoding()`.

`bindquotes(loose|strict|nobind)` specifies how `import delimited` handles double quotes in data. Specifying `loose` (the default) tells `import delimited` that it must have a matching open and closed double quote on the same line of data. `strict` tells `import delimited` that once it finds one double quote on a line of data, it should keep searching through the data for the matching double quote even if that double quote is on another line. Specifying `nobind` tells `import delimited` to ignore double quotes for binding.

`stripquotes(yes|no|default)` tells `import delimited` how to handle double quotes. `yes` causes all double quotes to be stripped. `no` leaves double quotes in the data unchanged. `default` automatically strips quotes that can be identified as binding quotes. `default` also will identify two adjacent double quotes as a single double quote because some software encodes double quotes that way.

`delimiters("`*chars*`"[, collapse|asstring])` allows you to specify other separation characters. For instance, if values in the file are separated by a semicolon, specify `delimiters(";")`. By default, `import delimited` will check if the file is delimited by tabs or commas based on the first line of data. Specify `delimiters("\t")` to use a tab character, or specify `delimiters("whitespace")` to use whitespace as a delimiter.

`collapse` forces `import delimited` to treat multiple consecutive delimiters as just one delimiter.

`asstring` forces `import delimited` to treat *chars* as one delimiter. By default, each character in *chars* is treated as an individual delimiter.

`parselocale(`*locale*`)` specifies the locale to use for interpreting numbers in the text file being imported. This option invokes an alternative parsing method and can result in slightly different behavior than not specifying this option. The default is to not use a locale when parsing numbers where the behavior is to treat `.` as the decimal separator. A list of available locales can be found at https://www.oracle.com/technetwork/java/javase/java8locales-2095355.html.

`decimalseparator(`*character*`)` specifies the character to use for interpreting the decimal separator when parsing numbers. This option implicitly invokes option `parselocale()` with your system's default locale. `parselocale(`*locale*`)` can be specified to override the default system locale.

`groupseparator(`*character*`)` specifies the character to use for interpreting the grouping separator when parsing numbers. This option implicitly invokes option `parselocale()` with your system's default locale. `parselocale(`*locale*`)` can be specified to override the default system locale.

`maxquotedrows(#|unlimited)` specifies the number of rows allowed inside a quoted string when parsing the file to import. The default is `maxquotedrows(20)`. If this option is specified without `bindquote(strict)`, then `maxquotedrows()` will be ignored.

Option `maxquotedrows(0)` is a synonym for `maxquotedrows(unlimited)`.

`numericcols(`*numlist*`|_all)` forces the data type of the column numbers in *numlist* to be numeric. Specifying `_all` will import all data as numeric.

`stringcols(`*numlist*`|_all)` forces the data type of the column numbers in *numlist* to be string. Specifying `_all` will import all data as strings.

`clear` specifies that it is okay to replace the data in memory, even though the current data have not been saved to disk.

The following option is available with `import delimited` but is not shown in the dialog box:

`favorstrfixed` forces `import delimited` to favor storing strings as a `str#`.

By default, `import delimited` will attempt to save space by importing string data as a `strL` if doing so will save space. The `favorstrfixed` option prevents the space-saving calculation from occurring, causing strings to be stored as a `str#` unless the string is larger than a `str#` can hold. In that case, `strL` must be used. See [R] **Limits** for details about the maximum size of a `str#`.

Options for export delimited

`delimiter("`*char*`"|tab)` allows you to specify other separation characters. For instance, if you want the values in the file to be separated by a semicolon, specify `delimiter(";")`. The default delimiter is a comma.

`delimiter(tab)` specifies that a tab character be used as the delimiter.

`novarnames` specifies that variable names not be written in the first line of the file; the file is to contain data values only.

`nolabel` specifies that the numeric values of labeled variables be written into the file rather than the label associated with each value.

`datafmt` specifies that all variables be exported using their display format. For example, the number 1000 with a display format of `%4.2f` would export as `1000.00`, not `1000`. The default is to use the raw, unformatted value when exporting.

`quote` specifies that string variables always be enclosed in double quotes. The default is to only double quote strings that contain spaces or the delimiter.

`replace` specifies that *filename* be replaced if it already exists.

Remarks and examples

stata.com

Remarks are presented under the following headings:

Introduction

`import delimited` reads into memory a text file in which there is one observation per line and the values are separated by commas, tabs, or some other delimiter. The two most common types of text data to import are comma-separated values (`.csv`) text files and tab-separated text files, often `.txt` files. `import delimited` will automatically detect either a comma or a tab as the delimiter.

Similarly, `export delimited` writes Stata data to a text file. By default, `export delimited` uses a comma as the delimiter, but you may specify another delimiter.

Imported string data containing ASCII or UTF-8 will always display correctly in the Data Editor and Results window. Imported string data containing extended ASCII may not display correctly unless you specify the character encoding using the `encoding()` option to convert the extended ASCII to UTF-8.

Exported text files are UTF-8 encoded.

If you are not sure that `import delimited` will do what you are looking for, see [D] **import** and [U] **22 Entering and importing data** for information about Stata's other commands for importing data.

Importing a text file

Suppose we have a `.csv` data file such as the following `auto.csv`, which contains variable names and data for different cars.

```
. copy https://www.stata.com/examples/auto.csv auto.csv
. type auto.csv
make,price,mpg,rep78,foreign
"AMC Concord",4099,22,3,"Domestic"
"AMC Pacer",4749,17,3,"Domestic"
"AMC Spirit",3799,22,,"Domestic"
"Buick Century",4816,20,3,"Domestic"
"Buick Electra",7827,15,4,"Domestic"
"Buick LeSabre",5788,18,3,"Domestic"
"Buick Opel",4453,26,,"Domestic"
"Buick Regal",5189,20,3,"Domestic"
"Buick Riviera",10372,16,3,"Domestic"
"Buick Skylark",4082,19,3,"Domestic"
```

We would like to import these data into Stata for subsequent analysis.

▷ Example 1: Importing all data

To import the complete dataset, we need to specify only the filename. `import delimited` assumes an extension of `.csv`. If our data were stored in a `.txt` file instead, we would need to specify the file extension. Here we enclose `auto` in double quotes (`" "`). We do this to remind you to use quotes for filenames with spaces, but it is not necessary here.

```
. import delimited "auto"
(5 vars, 10 obs)
```

We can verify that our data loaded correctly by using `list` or `browse`.

```
. list
```

	make	price	mpg	rep78	foreign
1.	AMC Concord	4099	22	3	Domestic
2.	AMC Pacer	4749	17	3	Domestic
3.	AMC Spirit	3799	22	.	Domestic
4.	Buick Century	4816	20	3	Domestic
5.	Buick Electra	7827	15	4	Domestic
6.	Buick LeSabre	5788	18	3	Domestic
7.	Buick Opel	4453	26	.	Domestic
8.	Buick Regal	5189	20	3	Domestic
9.	Buick Riviera	10372	16	3	Domestic
10.	Buick Skylark	4082	19	3	Domestic

Notice that `import delimited` automatically assigned the variable names such as `make` and `price` based on the first row of the data. If the variable names were located on, for example, line 3, we would have specified `varnames(3)`, and `import delimited` would have ignored the first two rows. If our file did not contain any variable names, we would have specified `varnames(nonames)`.

◁

▷ Example 2: Importing a subset of the data

`import delimited` also allows you to import a subset of the text data by using the `rowrange()` and `colrange()` options. Use `rowrange()` to specify which observations you want to import and `colrange()` to specify which variables you want to import.

Suppose that we want only cars that were manufactured by AMC. We can use the `drop` command to drop the cars manufactured by Buick after we import the data. If we know the rows in which AMC cars are located, we can also restrict our import to just those rows. Because `foreign` is constant, we also want to skip the last column.

To import rows 1 through 3 of the data in `auto.csv`, we need to specify `rowrange(2:4)` because the first row of the file contains the variable names. To import the first four columns, we need to also specify `colrange(1:4)`.

```
. clear
. import delimited "auto", rowrange(2:4) colrange(1:4)
(4 vars, 3 obs)
. list
```

	make	price	mpg	rep78
1.	AMC Concord	4099	22	3
2.	AMC Pacer	4749	17	3
3.	AMC Spirit	3799	22	.

`import delimited` still used the first line of the file to obtain the variable names even though we did not start our `rowrange()` specification with 1. `rowrange()` controls only which rows are read as data to be imported into Stata.

◁

Using other delimiters

Many delimited files use commas or tabs; other common delimiters are semicolons and whitespace. `import delimited` detects commas and tabs by default but can handle other characters. Suppose that you had the `auto.txt` file, which contains the following data.

```
"AMC Concord"   4099   22  3   "Domestic"
"AMC Pacer"     4749   17  3   "Domestic"
"AMC Spirit"    3799   22  NA  "Domestic"
"Buick Century" 4816   20  3   "Domestic"
"Buick Electra" 7827   15  4   "Domestic"
"Buick LeSabre" 5788   18  3   "Domestic"
"Buick Opel"    4453   26  NA  "Domestic"
"Buick Regal"   5189   20  3   "Domestic"
"Buick Riviera" 10372  16  3   "Domestic"
"Buick Skylark" 4082   19  3   "Domestic"
```

These data are whitespace delimited. If you use `import delimited` without any options, you will not get the results you expect.

```
. clear
. import delimited "auto.txt"
(1 var, 10 obs)
```

When `import delimited` tries to read data that have no tabs or commas, it is fooled into thinking that the data contain just one variable.

▷ Example 3: Changing the delimiter

We can use the `delimiters()` option to import the data correctly. `delimiters(" ")` tells `import delimited` to use spaces (" ") as the delimiter. Adding the `collapse` suboption will treat multiple consecutive space delimiters as one delimiter.

```
. clear
. import delimited "auto.txt", delimiters(" ", collapse)
(5 vars, 10 obs)
. describe
Contains data
  obs:            10
 vars:             5
-------------------------------------------------------------------------------
              storage   display    value
variable name   type    format     label      variable label
-------------------------------------------------------------------------------
v1              str13   %13s
v2              int     %8.0g
v3              byte    %8.0g
v4              str2    %9s
v5              str8    %9s
-------------------------------------------------------------------------------
Sorted by:
     Note: Dataset has changed since last saved.
```

The data that were imported now contain the correct number of variables and observations.

Because `import delimited` did not find variable names in the first row of `auto.txt`, Stata assigned default names of `v#` to the imported variables. If we wanted to specify our own names, we could have instead submitted

```
. clear
. import delimited make price mpg rep78 foreign using auto.txt,
> delimiters(" ", collapse)
(5 vars, 10 obs)
```

◁

Specifying variable types

The data in a file may contain a combination of string and numeric variables. `import delimited` will generally determine the correct data type for each variable. However, you may want to force a different data type by using the `numericcols()` or `stringcols()` option. For example, string values may be used to indicate missing values in a numeric variable, or you may want to import numeric values as strings to preserve leading zeros.

Another common case where you want to control the import type is when your data contain identifiers or other large numeric values. In this case, you should specify the `asdouble` option to avoid introducing duplicate values or losing values after the import.

▷ Example 4: Specify the storage type

Continuing with example 3, we know that the fourth variable, `rep78`, should be a numeric variable. But it was imported as a string because the value `NA` was used for missing values.

```
. list
```

	make	price	mpg	rep78	foreign
1.	AMC Concord	4099	22	3	Domestic
2.	AMC Pacer	4749	17	3	Domestic
3.	AMC Spirit	3799	22	NA	Domestic
4.	Buick Century	4816	20	3	Domestic
5.	Buick Electra	7827	15	4	Domestic
6.	Buick LeSabre	5788	18	3	Domestic
7.	Buick Opel	4453	26	NA	Domestic
8.	Buick Regal	5189	20	3	Domestic
9.	Buick Riviera	10372	16	3	Domestic
10.	Buick Skylark	4082	19	3	Domestic

To force rep78 to have a numeric storage type, we can use the numericcols(4) option.

```
. clear
. import delimited make price mpg rep78 foreign using "auto.txt",
> delimiters(" ", collapse) numericcols(4)
(5 vars, 10 obs)
. describe
Contains data
  obs:            10
 vars:             5
```

variable name	storage type	display format	value label	variable label
make	str13	%13s		
price	int	%8.0g		
mpg	byte	%8.0g		
rep78	int	%8.0g		
foreign	str8	%9s		

```
Sorted by:
     Note: Dataset has changed since last saved.
. list
```

	make	price	mpg	rep78	foreign
1.	AMC Concord	4099	22	3	Domestic
2.	AMC Pacer	4749	17	3	Domestic
3.	AMC Spirit	3799	22	.	Domestic
4.	Buick Century	4816	20	3	Domestic
5.	Buick Electra	7827	15	4	Domestic
6.	Buick LeSabre	5788	18	3	Domestic
7.	Buick Opel	4453	26	.	Domestic
8.	Buick Regal	5189	20	3	Domestic
9.	Buick Riviera	10372	16	3	Domestic
10.	Buick Skylark	4082	19	3	Domestic

rep78 is now stored as an int variable, and the NA values are replaced by ., the system missing value for numeric variables.

◁

Exporting to a text file

export delimited creates text files from the Stata dataset in memory. A comma-separated .csv file is created by default, but you can change the delimiter by specifying the delimiter() option and the file extension by specifying it with the *filename*.

▷ Example 5: Export all data

We want to export the data from example 4 to `myauto.csv`. We can use the `type` command to see the contents of the file.

```
. export delimited "myauto"
file myauto.csv saved
. type "myauto.csv"
make,price,mpg,rep78,foreign
AMC Concord,4099,22,3,Domestic
AMC Pacer,4749,17,3,Domestic
AMC Spirit,3799,22,,Domestic
Buick Century,4816,20,3,Domestic
Buick Electra,7827,15,4,Domestic
Buick LeSabre,5788,18,3,Domestic
Buick Opel,4453,26,,Domestic
Buick Regal,5189,20,3,Domestic
Buick Riviera,10372,16,3,Domestic
Buick Skylark,4082,19,3,Domestic
```

◁

▷ Example 6: Export a subset of the data

You can also export a subset of the data in memory by typing a variable list, specifying an `if` condition, specifying a range with an `in` condition, or a combination of the three. For example, here we export only the first 5 observations of the `make`, `mpg`, and `rep78` variables.

```
. export delimited make mpg rep78 in 1/5 using "myauto", replace
file myauto.csv saved
```

If you open `myauto.csv`, you will see that only the 5 observations shown in example 5 appear in the file. We specified the `replace` option because we previously exported data to `myauto.csv`. If we had not specified `replace`, we would have received an error message.

◁

Video example

Importing delimited data

Stored results

`import delimited` stores the following in `r()`:

Scalars

`r(N)`	number of observations imported
`r(k)`	number of variables imported

Also see

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

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