22 Entering and importing data

Contents

22.1 Overview

To enter or import data into Stata, you can use the following:

[D] **edit** and [D] **input**	enters data from the keyboard
[D] **import delimited**	reads delimited text data
[D] **import excel**	reads Excel files
[D] **import sas**	reads SAS files
[D] **import sasxport5**	reads data in SAS XPORT Version 5 format
[D] **import sasxport8**	reads data in SAS XPORT Version 8 format
[D] **import spss**	reads SPSS files
[D] **infile (free format)**	reads unformatted text data
[D] **infile (fixed format)** or [D] **infix (fixed format)**	reads formatted text data
[D] **infile (fixed format)**	reads EBCDIC data
[D] **odbc**	reads from an ODBC source
[D] **jdbc**	reads from a JDBC source
[D] **import fred**	reads Federal Reserve Economic Data
[D] **import haver**	reads data from Haver Analytics databases
[D] **import haverdirect**	reads data from Haver Analytics cloud servers
[D] **import dbase**	reads dBase files
[SP] **spshape2dta**	converts shapefiles into a form Stata can use

Because dataset formats differ, you should familiarize yourself with each method.

[D] **infile (fixed format)** and [D] **infix (fixed format)** are two different commands that do the same thing. Read about both, and then use whichever appeals to you.

Alternatively, `edit` and `input` both allow you to enter data from the keyboard. `edit` opens a Data Editor, and `input` allows you to type at the command line.

After you have read this chapter, also see [D] **import** for more examples of the different commands to input data.

❑ Technical note

Strings in Stata are stored in UTF-8 format, the most common string storage format across software packages. You probably do not need to take any special steps when importing strings from other packages. However, if you are importing data with strings that are stored as extended ASCII, including extended ASCII strings in Stata 13 and earlier datasets, you need to convert those strings to UTF-8. You will know whether you have extended ASCII strings that need conversion, because if you do, you will not see the characters you expect in your strings after you import them. Stata provides the command `unicode translate` to help you. See [D] **unicode translate**, [U] **12.4.2 Handling Unicode strings**, and [D] **unicode** for more information.

❑

22.2 Determining which method to use

Below are several rules that, when applied sequentially, will direct you to the appropriate method for entering your data. After the rules is a description of each command, as well as a reference to the corresponding entry in the *Reference* manuals.

1. If you have a few data and simply wish to type the data directly into Stata at the keyboard, see [D] **edit**—doing so should be easy. Also see [D] **input**.
2. If your dataset is in binary format or the internal format of some software package, you have several options:
    a. If the data are in a spreadsheet, copy and paste the data into Stata's Data Editor; see [D] **edit** for details.
    b. If the data are in an Excel spreadsheet, use `import excel` to read them; see [D] **import excel**.
    c. If the data are in a SAS file, use `import sas` to read the data; see [D] **import sas**.
    d. If the data are in SAS XPORT Version 5 or Version 8 format, use `import sasxport5` or `import sasxport8` to read the data; see [D] **import sasxport5** and [D] **import sasxport8**.
    e. If the data are in an SPSS file, use `import spss` to read the data; see [D] **import spss**.
    f. If you wish to import data from the online Federal Reserve Economic Data (FRED) database, use `import fred`; see [D] **import fred**.
    g. If the data are in a Haver Analytics database on your local network (Haver Analytics provides economics and financial databases), and you are using Stata for Windows, use `import haver` to read the data; see [D] **import haver**.
    h. If the data are in a Haver Analytics cloud database and you are using Stata for Windows, use `import haverdirect` to read the data; see [D] **import haverdirect**.
    i. If the data are in a dBase file, use `import dbase`; see [D] **import dbase**.
    j. Translate the data into text format by using the other software. For instance, in most software, you can save data as tab-delimited or comma-separated text. Then, see [D] **import delimited**.
    k. If the data are located in an ODBC source, which typically includes databases and spreadsheets, you can use the `odbc load` command to import the data; see [D] **odbc**.

l. If the data are located in a database and the database vendor has a JDBC driver, you can use the `jdbc load` command to import the data; see [D] **jdbc**.

m. If you wish to use shapefile data with Stata, use `spshape2dta` to convert it to a form Stata can use; see [SP] **spshape2dta**.

n. Other software packages are available that will convert non–Stata format data files into Stata-format files.

3. If the dataset has one observation per line and the data are tab- or comma separated, use `import delimited`; see [D] **import delimited**. This is the easiest way to read text data.
4. If the dataset is formatted and that formatting information is required to interpret the data, you can use `infile` with a dictionary or `infix`; see [D] **infile (fixed format)** or [D] **infix (fixed format)**.
5. If there are no string variables, you can use `infile` without a dictionary: see [D] **infile (free format)**.
6. If all the string variables in the data are enclosed in (single or double) quotes, you can use `infile` without a dictionary; see [D] **infile (free format)**.
7. If the string variables have no blanks and are whitespace-delimited, you can use `infile` without a dictionary; see [D] **infile (free format)**.
8. If the data are in EBCDIC format, see [D] **infile (fixed format)**.
9. If you make it to here, see [D] **infile (fixed format)** or [D] **infix (fixed format)**.

22.2.1 Entering data interactively

If you have a few data, you can type the data directly into Stata; see [D] **edit** or [D] **input**. Otherwise, we assume that your data are stored on disk.

22.2.2 Copying and pasting data

If your data are in another program and you wish to analyze them with Stata, first see if the program you are using allows you to copy the data to the clipboard. If it does, do so, and then open the Data Editor in Stata and select **Edit > Paste** to paste the data into Stata.

22.2.2.1 Video example

Copy/paste data from Excel into Stata

22.2.3 If the dataset is in binary format

Stata can read text datasets, which is technical jargon for datasets composed of characters—datasets that can be typed on your screen or printed on your printer. The alternative, binary datasets, can only sometimes be read by Stata. Binary datasets are popular, and almost every software package has its own binary format. Stata `.dta` datasets are an example of a binary format that Stata can read. The Excel `.xls` and `.xlsx` formats are other binary formats that Stata can read. The OpenOffice `.ods` format is a binary format that Stata cannot read.

If your dataset is in binary format or in the internal format of another software package that Stata cannot import, you must translate it into plain text or use some other program for conversion to Stata format. If this dataset is an Excel `.xls` or `.xlsx` file, you can read it by using Stata's `import excel` command; see [D] **import excel**. If this dataset is located in a database or an ODBC source, see **[U] 22.4 ODBC sources**. If this dataset is located in a database and the database vendor has a JDBC driver, see **[U] 22.5 JDBC sources**. If the dataset is in SAS format, you can read it by using `import sas`. If the data are in SAS XPORT Version 5 format or in SAS XPORT Version 8 format, you can read the data by using Stata's `import sasxport5` or `import sasxport8` command; see [D] **import sasxport5** and [D] **import sasxport8**. You can read data in SPSS `.sav` format by using `import spss`; see [D] **import spss**. If the data are available via the Federal Reserve Economic Data (FRED) online database, you can read the data by using Stata's `import fred` command; see [D] **import fred**. If the dataset is in a Haver Analytics database on your local network, you can read it by using Stata's `import haver` command; see [D] **import haver**. If the dataset is in a Haver Analytics cloud database, you can read it by using Stata's `import haverdirect` command; see [D] **import haverdirect**. If the dataset is in dBase format, you can read it by using Stata's `import dbase` command; see [D] **import dbase**. If you have a shapefile and wish to use it with Stata, use `spshape2dta` to convert it to a form that can be used with Stata; see [SP] **spshape2dta**. If the dataset is in EBCDIC format, you can read it by using Stata's `infile` command; see [D] **infile (fixed format)**.

Detecting whether data are stored in binary format can be tricky. For instance, many Windows users wish to read data that have been entered into a word processor—let's assume Word. Unwittingly, they have stored the dataset as a Word document. The dataset looks like text to them: When they look at it in Word, they see readable characters. The dataset seems to even pass the printing test in that Word can print it. Nevertheless, the dataset is not text; it is stored in an internal Word format, and the data cannot really pass the printing test because only Word can print it. To read the dataset, Windows users must use it in Word and then store it as a plain text (`.txt`) file.

So, how do you know whether your dataset is binary? Here's a simple test: regardless of the operating system you use, start Stata and type `type` followed by the name of the file:

```
. type myfile.raw
output will appear
```

You do not have to list the entire file; press *Break* when you have seen enough.

Do you see things that look like hieroglyphics? If so, the dataset is binary.

If it looks like data, however, the file is (probably) plain text.

Let's assume that you have a text dataset that you wish to read. The data's format will determine the command you need to use. The different formats are discussed in the following sections.

22.2.4 If the data are simple

The easiest way to read text data is with `import delimited`; see [D] **import delimited**.

`import delimited` is smart: it looks at the dataset, determines what it contains, and then reads it. That is, `import delimited` is smart given certain restrictions, such as that the dataset has one observation per line and that the values are tab- or comma separated. `import delimited` can read this

begin data1.csv

```
M,Joe Smith,288,14
M,K Marx,238,12
F,Farber,211,7
```

end data1.csv

or this (which has variable names on the first line)

begin data2.csv
```
sex, name, dept, division
M,Joe Smith,288,14
M,K Marx,238,12
F,Farber,211,7
```
end data2.csv

or this (which has one tab character separating the values):

begin data3.txt
```
M       Joe Smith       288     14
M       K Marx  238     12
F       Farber  211     7
```
end data3.txt

This looks odd because of how tabs work; `data3.txt` could similarly have a variable header, but `import delimited` cannot read

begin data4.txt
```
M       Joe Smith       288     14
M       K Marx          238     12
F       Farber          211     7
```
end data4.txt

which has spaces rather than tabs.

There is a way to tell `data3.txt` from `data4.txt`: Ask Stata to type the data and show the tabs by typing

```
. type data3.txt, showtabs
M<T>Joe Smith<T>288<T>14
M<T>K Marx<T>238<T>12
F<T>Farber<T>211<T>7
. type data4.txt, showtabs
M       Joe Smith       288     14
M       K Marx          238     12
F       Farber          211     7
```

22.2.5 If the dataset is formatted and the formatting is significant

If the dataset is formatted and formatting information is required to interpret the data, see [D] **infile (fixed format)** or [D] **infix (fixed format)**.

Using `infix` or `infile` with a data dictionary is something new users want to avoid if at all possible.

The purpose of this section is only to take you to the most complicated of all cases if there is no alternative. Otherwise, you should wait and see if it is necessary. Do not misinterpret this section and say, "Ah, my dataset is formatted, so at last I have a solution."

Just because a dataset is formatted does not mean that you have to exploit the formatting information. The following dataset is formatted

begin data5.raw
```
1   27.39    12
2    1.00     4
3  100.10   100
```
end data5.raw

in that the numbers line up in neat columns, but you do not need to know the information to read it. Alternatively, consider the same data run together:

begin data6.raw
```
1 27.39 12
2  1.00  4
3100.10100
```
end data6.raw

This dataset is formatted, too, and you must know the formatting information to make sense of "3100.10100". You must know that variable 2 starts in column 4 and is six characters long to extract the 100.10. It is datasets like `data6.raw` that you should be looking for at this stage—datasets that make sense only if you know the starting and ending columns of data elements. To read data such as `data6.raw`, you must use either `infix` or `infile` with a data dictionary.

Reading unformatted data is easier. If you need the formatting information to interpret the data, then you must communicate that information to Stata, which means that you will have to type it. This is the hardest kind of data to read, but Stata can do it. See [D] **infile (fixed format)** or [D] **infix (fixed format)**.

Looking back at `data4.raw`,

begin data4.raw
```
M       Joe Smith       288     14
M       K Marx          238     12
F       Farber          211     7
```
end data4.raw

you may be uncertain whether you have to read it with a data dictionary. If you are uncertain, do not jump yet.

Finally, here is an obvious example of unformatted data:

begin data7.raw
```
1 27.39          12
2 1 4
3 100.1 100
```
end data7.raw

Here blanks separate one data element from the next and, in one case, many blanks, although there is no special meaning attached to more than one blank.

The following sections discuss datasets that are unformatted or formatted in a way that do not require a data dictionary.

22.2.6 If there are no string variables

If there are no string variables, see [D] **infile (free format)**.

Although the dataset `data7.raw` is unformatted, it can still be read using `infile` without a dictionary. This is not the case with `data4.raw` because this dataset contains undelimited string variables with embedded blanks.

❑ Technical note

Some Stata users prefer to read data with a data dictionary, even when we suggest differently, as above. They like the convenience of the data dictionary—they can sit in front of an editor and carefully compose the list of variables and attach variable labels rather than having to type the variable list (correctly) on the Stata command line. However, they can create a do-file containing the `infile` statement and thus have all the advantages of a data dictionary without some of the (extremely technical) disadvantages of data dictionaries.

Nevertheless, we do tend to agree with such users—we, too, prefer data dictionaries. Our recommendations, however, are designed to work in all cases. If the dataset is unformatted and contains no string variables, it can always be read without a data dictionary, whereas only sometimes can it be read with a data dictionary.

The distinction is that `infile` without a data dictionary performs stream I/O, whereas with a data dictionary it performs record I/O. The difference is intentional—it guarantees that you will be able to read your data into Stata somehow. Some datasets require stream I/O, others require record I/O, and still others can be read either way. Recommendations 1–5 identify datasets that either require stream I/O or can be read either way.

❑

We are now left with datasets that contain at least one string variable.

22.2.7 If all the string variables are enclosed in quotes

If all the string variables in the data are enclosed in (single or double) quotes, see [D] **infile (free format)**.

See [U] **24 Working with strings** for a formal definition of strings, but as a quick guide, a string variable is a variable that takes on values like "`bob`" or "`joe`", as opposed to numeric variables that take on values like 1, 27.5, and –17.393. Undelimited strings—strings not enclosed in quotes—can be difficult to read.

Here is an example including delimited string variables:

begin data8.raw
```
"M" "Joe Smith" 288 14
"M" "K Marx" 238 12
"F" "Farber" 211 7
```
end data8.raw

or

begin data8.raw, alternative format
```
"M" "Joe Smith" 288  14
"M" "K Marx"    238  12
"F" "Farber"    211   7
```
end data8.raw, alternative format

Both of these are merely variations on `data4.raw` except that the strings are enclosed in quotes. Here `infile` without a dictionary can be used to read the data.

Here is another version of `data4.raw` without delimiters or even formatting:

begin data9.raw
```
M Joe Smith 288 14
M K Marx 238 12
F Farber 211 7
```
end data9.raw

What makes these data difficult? Blanks sometimes separate values and sometimes are nothing more than a blank within a string. For instance, you cannot tell whether Farber has first initial F with missing sex or is instead female with a missing first initial.

Fortunately, such data rarely happen. Either the strings are delimited, as we showed in `data8.raw`, or the data are in columns, as in `data4.raw`.

22.2.8 If the undelimited strings have no blanks

There is a case in which uncolumnized, undelimited strings cause no confusion — when they contain no blanks. For instance, if our data contained only last names,

begin data10.raw

```
Smith 288 14
Marx 238 12
Farber 211 7
```

end data10.raw

Stata could read it without a data dictionary. Caution: the last names must contain no blanks — no Van Owen's or von Beethoven's.

If the undelimited string variables have no blanks, see [D] **infile (free format)**.

22.2.9 If you have EBCDIC data

You may rarely encounter data from a mainframe that is encoded in extended binary coded decimal interchange code (EBCDIC). EBCDIC is used on some IBM mainframe operating systems.

If you have EBCDIC data, you should have information on that data specifying where each field begins and ends and what type of data is in that field. You can read EBCDIC data in the same way that you read fixed-format text data, using `infile` (see [D] **infile (fixed format)**. You create a data dictionary that tells Stata which columns to read for each field, and you merely specify the `ebcdic` option with the `infile` command to read the data.

Alternatively, you can convert an EBCDIC file to an ASCII text file with the `filefilter` command. See [D] **filefilter**.

22.2.10 If you make it to here

If you make it to here, see [D] **infile (fixed format)** or [D] **infix (fixed format)**.

Remember `data4.raw`?

begin data4.raw

```
M       Joe Smith       288     14
M       K Marx          238     12
F       Farber          211     7
```

end data4.raw

It can be read using either `infile` with a dictionary or `infix`.

22.3 If you run out of memory

You may need to tweak a setting; see **[U] 6 Managing memory** and [D] **memory**.

You can also try to conserve memory.

When you read the data, did you specify variable types? Stata can store integers more compactly than floats and small integers more compactly than large integers; see **[U] 12 Data**.

If that is not sufficient, you will have to resort to reading the data in pieces. Both `infile` and `infix` allow you to specify an `in` *range* qualifier, and, here the range is interpreted as the observation range to read. Thus, `infile ... in 1/100` would read observations 1–100 of your data and stop.

`infile ... in 101/200` would read observations 101–200. The end of the range may be specified as larger than the actual number of observations in the data. If the dataset contained only 150 observations, `infile ... in 101/200` would read observations 101–150.

Another way of reading the data in pieces is to specify the `if` *exp* qualifier. Say that your data contained an equal number of males and females, coded as the variable `sex` (which you will read) being 0 or 1, respectively. You could type `infile ... if sex==0` to read the males. `infile` will read an observation, determine if `sex` is zero, and if not, throw the observation away. You could read just the females by typing `infile ... if sex==1`.

If the dataset is really big, perhaps you need only a random sample of the data—you never intended to analyze the entire dataset. Because `infile` and `infix` allow `if` *exp*, you could type `infile ... if runiform()<.1`. `runiform()` is the uniformly distributed random-number generator; see [FN] **Random-number functions**. This method would read an approximate 10% sample of the data. If you are serious about using random samples, do not forget to set the seed before using `runiform()`; see [R] **set seed**.

The final approach is to read all the observations but only some of the variables. When reading data without a data dictionary, you can specify `_skip` for variables, indicating that the variable is to be skipped. When reading with a data dictionary or using `infix`, you can specify the actual columns to read, skipping any columns you wish to ignore.

If you are using `import excel`, you can read a subset of an Excel worksheet by using the `cellrange()` option. See [D] **import excel**.

22.4 ODBC sources

If your dataset is located in a network database or shared spreadsheet, you may be able to import your data via ODBC. Open Database Connectivity (ODBC) is a standard for exchanging data between programs. Stata supports the ODBC standard for importing data via the `odbc` command and can read from any ODBC source on your computer.

This process requires a data source, such as a database located on a network. To use the `odbc` command to import data from a database requires that the database first be set up as an ODBC source on the same machine that is running Stata. The database itself does not have to be on the same machine, just the definition of that database as the ODBC source. On a Windows machine, an ODBC source is added via a Control Panel called "Data Sources". Also, typing `odbc list` from Stata displays all the ODBC sources that are provided by the computer.

If the database is functioning and the appropriate data source has been set up on the same machine as Stata, one call using `odbc load` is all that is needed to import data. For a more thorough description of this process, see [D] **odbc**.

22.5 JDBC sources

If your dataset is located in a database, you may be able to import your data via JDBC. Java Database Connectivity (JDBC) is a standard for exchanging data between programs. Stata supports the JDBC standard for importing data from relational databases or nonrelational database-management systems that have rectangular data.

Using the `jdbc` command to import data from a database requires that the database vendor supply a JDBC driver for you to download and install. If the database is functioning and the driver can be found by Stata, one call using `jdbc load` is all that is needed to import data. For a more thorough description of this process, see [D] **jdbc**.

For suggested citations, see the FAQ on citing Stata documentation.