# mi set — Declare multiple-imputation data

## Description

`mi set` is used to set a regular Stata dataset to be an `mi` dataset. `mi set` is also used to modify the attributes of an already set dataset. An `mi set` dataset has the following attributes:

- The data are recorded in a *style*: `wide`, `mlong`, `flong`, or `flongsep`; see [MI] **Styles**.
- Variables are registered as `imputed`, `passive`, or `regular`, or they are left unregistered.
- In addition to $m = 0$, the data with missing values, the data include $M \geq 0$ imputations of the imputed variables.

`mi set` *style* begins the setting process by setting the desired style. `mi set` *style* sets all variables as unregistered and sets $M = 0$.

`mi register` registers variables as `imputed`, `passive`, or `regular`. Variables can be registered one at a time or in groups and can be registered and reregistered.

`mi unregister` unregisters registered variables, which is useful if you make a mistake. Exercise caution. Unregistering an imputed or passive variable can cause loss of the filled-in missing values in $m > 0$ if your data are recorded in the wide or mlong styles. In such cases, just `mi register` the variable correctly without `mi unregistering` it first.

`mi set M` modifies $M$, the total number of imputations. $M$ may be increased or decreased. $M$ may be set before or after imputed variables are registered.

`mi set m` drops selected imputations from the data.

`mi unset` is a rarely used command to unset the data. Better alternatives include `mi extract` and `mi export` (see [MI] **mi extract** and [MI] **mi export**, respectively).

## Menu

Statistics > Multiple imputation

## Syntax

```
mi set style

    where style is wide
                   mlong
                   flong
                   flongsep name


mi register {imputed|passive|regular} varlist

mi unregister varlist


mi set M {=|+=|-=} #

mi set m -= (numlist)


mi unset [, asis]
```

The mi suite of commands does not allow alias variables; see [D] **frunalias** for advice on how to get around this restriction.

## Option for mi unset

`asis` is rarely used and is intended for programmers. `mi unset, asis` unsets the `mi` data as is. This means that it will not rename the `_mi_*` system variables and the `_#_*` variables (wide `mi` style). By default, `mi unset` renames the `_mi_*` variables as `mi_*` and the `_#_*` variables as `*_#_`. Most `mi` commands, including `mi unset` without option `asis`, will not work without the `_mi_*` system variables. `mi unset, asis`, however, will work even without those variables (in case they are accidentally dropped); it will simply unset the data as is. The resulting dataset may not always be usable; for instance, without the `_mi_m` variable, you may not be able to identify imputations, and without the `_mi_id` variable, you may not be able to identify the imputed observations. You should exercise caution when using this option.

## Remarks and examples

stata.com

Data must be `mi set` before they can be used with the other `mi` commands. There are two ways data can become `mi set`: direct use of `mi set` *style* or use of `mi import` (see [MI] **mi import**).

The `mi register`, `mi set M`, and `mi set m` commands are for use with already set data and are useful even with imported data.

Remarks are presented under the following headings:

## mi set style

`mi set` *style* begins the setting process. `mi set` *style* has the following forms:

```
mi set wide
mi set mlong
mi set flong
mi set flongsep name
```

It does not matter which style you choose because you can always use `mi convert` (see [MI] **mi convert**) to change the style later. We typically choose `wide` to begin.

If your data are large, you may have to use `flongsep`. `mi set flongsep` requires you to specify a name for the flongsep dataset collection. See *Advice for using flongsep* in [MI] **Styles**.

If you intend to have super-varying variables, you need to choose either `flong` or `flongsep`, or you will need to `mi convert` to flong or flongsep style later.

The current style of the data is shown by the `mi query` and `mi describe` commands; see [MI] **mi describe**.

## mi register and mi unregister

`mi register` has three forms:

```
mi register imputed varlist
mi register passive varlist
mi register regular varlist
```

See [MI] **Glossary** for a definition of imputed, passive, and regular variables.

You are required to register imputed variables. If you intend to use `mi impute` (see [MI] **mi impute**) to impute missing values, you must still register the variables first.

Concerning passive variables, we recommend that you register them, and if your data are style wide, you are required to register them. If you create passive variables by using `mi passive` (see [MI] **mi passive**), that command automatically registers them for you.

Whether you register regular variables is up to you. Registering them is safer in all styles except wide, where it does not matter. We say registering is safer because regular variables should not vary across $m$, and in the long styles, you can unintentionally create variables that vary. If variables are registered, `mi` will detect and fix mistakes for you.

The names of imputation and passive variables may not exceed 29 characters. In the wide style, the names of these variables may be restricted to less than 29 characters depending on the number of imputations. In the flongsep style, the names of regular variables in addition to the names of imputation and passive variables also may not exceed 29 characters.

Super-varying variables—see [MI] **Glossary**—rarely occur, but if you have them, be aware that they can be stored only in flong and flongsep data and that they never should be registered.

The registration status of variables is listed by `mi describe` (see [MI] **mi describe**).

Use `mi unregister` if you accidentally register a variable incorrectly, with one exception: if you mistakenly register a variable as `imputed` but intended to register it as `passive`, or vice versa, use `mi register` directly to reregister the variable. The mere act of unregistering a passive or imputed variable can cause values in $m > 0$ to be replaced with those from $m = 0$ if the data are wide or mlong.

That exception aside, you first `mi unregister` variables before reregistering them.

## mi set M and mi set m

`mi set M` is seldom used, and `mi set m` is sometimes used.

`mi set M` sets $M$, the total number of imputations. The syntax is

```
mi set M  = #
mi set M += #
mi set M -= #
```

`mi set M = #` sets $M = \#$. Imputations are added or deleted as necessary. If imputations are added, the new imputations obtain their values of imputed and passive variables from $m = 0$, which means that the missing values are not yet replaced in the new imputations. It is not necessary to increase $M$ if you intend to use `mi impute` to impute values; see [MI] **mi impute**.

`mi set M += #` increments $M$ by #.

`mi set M -= #` decrements $M$ by #.

`mi set m -= (`*numlist*`)` deletes the specified imputations. For instance, if you had $M = 5$ imputations and wanted to delete imputation 2, leaving you with $M = 4$, you would type `mi set m -= (2)`.

## mi unset

If you wish to unset your data, your best choices are `mi extract` and `mi export`; see [MI] **mi extract** and [MI] **mi export**. The `mi extract 0` command replaces the data in memory with the data from $m = 0$, unset. The `mi export` command replaces the data in memory with unset data in a form that can be sent to a non–Stata user.

`mi unset` is included for completeness, and if it has any use at all, it would be by programmers.

## Also see

[MI] **Intro** — Introduction to mi

[MI] **mi convert** — Change style of mi data

[MI] **mi describe** — Describe mi data

[MI] **mi export** — Export mi data

[MI] **mi extract** — Extract original or imputed data from mi data

[MI] **mi import** — Import data into mi

[MI] **mi XXXset** — Declare mi data to be svy, st, ts, xt, etc.

[MI] **Styles** — Dataset styles

[D] **frunalias** — Change storage type of alias variables