**mi update** — Ensure that mi data are consistent

### Syntax

```
mi update
```

### Menu

Statistics > Multiple imputation

### Description

`mi update` verifies that `mi` data are consistent. If the data are not consistent, `mi update` reports the inconsistencies and makes the necessary changes to make the data consistent.

`mi update` can change the sort order of the data.

### Remarks and examples

Remarks are presented under the following headings:

- **Purpose of mi update**
- **What mi update does**
- **mi update is run automatically**

#### Purpose of mi update

`mi update` allows you to

- change the values of existing variables, whether imputed, passive, regular, or unregistered;
- add or remove missing values from imputed variables (or from any variables);
- drop variables;
- create new variables;
- drop observations; and
- duplicate observations (but not add observations in other ways).

You can make any or all of the above changes and then type

```
.mi update
```

and `mi update` will handle making whatever additional changes are required to keep the data consistent. For instance,

```
.drop if sex==1
(75 observations deleted)
.mi update
(375 m>0 obs. dropped due to dropped obs. in m=0)
```

In this example, we happen to have five imputations and are working with flongsep data. We dropped 75 observations in \( m = 0 \), and that still left \( 5 \times 75 = 375 \) observations to be dropped in \( m > 0 \).
mi update — Ensure that mi data are consistent

The messages mi update produces vary according to the style of the data because the changes required to make the data consistent are determined by the style. Had we been working with flong data, we might have seen

```
. drop if sex==1
(450 observations deleted)
. mi update
(system variable _mi_id updated due to change in number of obs.)
```

With flong data in memory, when we dropped if sex==1, we dropped all 75 + 5 × 75 = 450 observations, so no more observations needed to be dropped; but here mi update needed to update one of its system variables because of the change we made.

Had we been working with mlong data, we might have seen

```
. drop if sex==1
(90 observations deleted)
. mi update
(system variable _mi_id updated due to change in number of obs.)
```

The story here is very much like the story in the flong case. In mlong data, dropping if sex==1 drops the 75 observations in \( m = 0 \) and also drops the incomplete observations among the 75 in \( m = 1, m = 2, \ldots, m = 5 \). In this example, there are three such observations, so a total of \( 75 + 5 \times 3 = 90 \) were dropped, and because of the change, mi update needed to update its system variable.

Had we been using wide data, we might have seen

```
. drop if sex==1
(75 observations deleted)
. mi update
```

mi update’s silence indicates that mi update did nothing, because after dropping observations in wide data, nothing more needs to be done. We could have skipped typing mi update here, but do not think that way because changing values, dropping variables, creating new variables, dropping observations, or creating new observations can have unanticipated consequences.

For instance, in our data is variable farmincome, and it seems obvious that farmincome should be 0 if the person does not have a farm, so we type

```
. replace farmincome = 0 if !farm
(15 real changes made)
```

After changing values, you should type mi update even if you do not suspect that it is necessary. Here is what happens when we do that with these data:

```
. mi update
(12 m=0 obs. now marked as complete)
```

Typing mi update was indeed necessary! We forgot that the farmincome variable was imputed, and it turns out that the variable contained missing in 12 nonfarm observations; mi needed to deal with that.

Running mi update is so important that mi itself is constantly running it just in case you forget. For instance, let’s “forget” to type mi update and then convert our data to wide:

```
. replace farmincome = 0 if !farm
(15 real changes made)
. mi convert wide, clear
(12 m=0 obs. now marked as complete)
```
The parenthetical message was produced because mi convert ran mi update for us. For more information on this, see [MI] noupdate option.

What mi update does

- mi update checks whether you have changed $N$, the number of observations in $m = 0$, and resets $N$ if necessary.
- mi update checks whether you have changed $M$, the number of imputations, and adjusts the data if necessary.
- mi update checks whether you have added, dropped, registered, or unregistered any variables and takes the appropriate action.
- mi update checks whether you have added or deleted any observations. If you have, it then checks whether you carried out the operation consistently for $m = 0, m = 1, \ldots, m = M$. If you have not carried it out consistently, mi update carries it out consistently for you.
- In the mlong, flong, and flongsep styles, mi update checks system variable _mi_id, which links observations across $m$, and reconstructs the variable if necessary.
- mi update checks that the system variable _mi_miss, which marks the incomplete observations, is correct and, if not, updates it and makes any other changes required by the change.
- mi update verifies that the values recorded in imputed variables in $m > 0$ are equal to the values in $m = 0$ when they are nonmissing and updates any that differ.
- mi update verifies that the values recorded in passive variables in $m > 0$ are equal to the values recorded in $m = 0$’s complete observations and updates any that differ.
- mi update verifies that the values recorded in regular variables in $m > 0$ equal the values in $m = 0$ and updates any that differ.
- mi update adds any new variables in $m = 0$ to $m > 0$.
- mi update drops any variables from $m > 0$ that do not appear in $m = 0$.

mi update is run automatically

As we mentioned before, running mi update is so important that many mi commands simply run it as a matter of course. This is discussed in [MI] noupdate option. In a nutshell, the mi commands that run mi update automatically have a noupdate option, so you can identify them, and you can specify the option to skip running the update and so speed execution, but only with the adrenaline rush caused by a small amount of danger.

Whether you specify noupdate or not, we advise you to run mi update periodically and to always run mi update after dropping or adding variables or observations, or changing values.

Also see

[MI] intro — Introduction to mi
[MI] noupdate option — The noupdate option