Title

stata.com

byable — Make programs byable

Description Syntax Option Remarks and examples Also see

Description

Most Stata commands allow the use of the by prefix; see [D] **by**. For example, the syntax diagram for the regress command could be presented as

[by *varlist*:] regress ...

This entry describes the writing of programs (ado-files) so that they will allow the use of Stata's by *varlist*: prefix; see [D] **by**. If you take no special actions and write the program myprog, then by *varlist*: cannot be used with it:

```
. by foreign:  myprog
myprog may not be combined with by
r(190);
```

By reading this entry, you will learn how to modify your program so that by does work with it:

```
. by foreign:  myprog
-------------------------------------------------------------------------------
-> foreign = Domestic
   (output for first by-group appears)
-------------------------------------------------------------------------------
-> foreign = Foreign
   (output for first by-group appears)
. _
```

Syntax

program [define] *program_name*

[, ... byable(recall[, noheader] | onecall) ...]

Option

byable(recall[, noheader] | onecall) specifies that the program is to allow the by prefix to be used with it and specifies the style in which the program is coded.

There are two supported styles, known as byable(recall) and byable(onecall). byable(recall) programs are usually—not always—easier to write and byable(onecall) programs are usually—not always—faster.

byable(recall) programs are executed repeatedly, once per by group. byable(onecall) programs are executed only once and it is the program's responsibility to handle the implications of the by prefix if it is specified.

byable(recall, noheader) programs are distinguished from byable(recall) programs in that by will not display a by-group header before each calling of the program.

byable(onecall) programs are required to handle the by...: prefix themselves, including displaying the header should they wish that. See *Remarks and examples* for details.

Remarks and examples

stata.com

Remarks are presented under the following headings:

If you have not read [P] **sortpreserve**, please do so.

Programs that are written to be used with `by` *varlist*`:` are said to be "byable". Byable programs do not require the use of `by` *varlist*`:`; they merely allow it. There are two ways that programs can be made byable, known as `byable(recall)` and `byable(onecall)`.

`byable(recall)` is easy to use and is sufficient for programs that report the results of calculation (class-1 programs as defined in [P] **sortpreserve**). `byable(recall)` is the method most commonly used to make programs byable.

`byable(onecall)` is more work to program and is intended for use in all other cases (class-2 and class-3 programs as defined in [P] **sortpreserve**).

byable(recall) programs

Say that you already have written a program (ado-file) and that it works; it merely does not allow `by`. If your program reports the results of calculations (such as `summarize`, `regress`, and most of the other statistical commands), then probably all you have to do to make your program byable is add the `byable(recall)` option to its `program` statement. For instance, if your `program` statement currently reads

```
program myprog, rclass sortpreserve
        ...
end
```

change it to read

```
program myprog, rclass sortpreserve byable(recall)
        ...
end
```

The only change you should need to make is to add `byable(recall)` to the `program` statement. Adding `byable(recall)` will be the only change required if

- Your program leaves behind no newly created variables. Your program might create temporary variables in the midst of calculation, but it must not leave behind new variables for the user. If your program has a `generate()` option, for instance, some extra effort will be required.
- Your program uses `marksample` or `mark` to restrict itself to the relevant subsample of the data. If your program does not use `marksample` or `mark`, some extra effort will be required.

Here is how `byable(recall)` works: if your program is invoked with a `by` *varlist*`:` prefix, your program will be executed K times, where K is the number of by-groups formed by the by-variables. Each time your program is executed, `marksample` will know to mark out the observations that are not being used in the current by-group.

Therein is the reason for the two guidelines on when you need to include only `byable(recall)` to make `by` *varlist*`:` work:

- If your program creates permanent, new variables, then it will create those variables when it is executed for the first by-group, meaning that those variables will already exist when it is executed for the second by-group, causing your program to issue an error message.
- If your program does not use `marksample` to identify the relevant subsample of the data, then each time it is executed, it will use too many observations—it will not honor the by-group—and will produce incorrect results.

There are ways around both problems, and here is more than you need:

function `_by()`	takes no arguments; returns 0 when program is not being by'd; returns 1 when program is being by'd.
function `_byindex()`	takes no arguments; returns 1 when program is not being by'd; returns 1, 2, ... when by'd and 1st call, 2nd call, ...
function `_bylastcall()`	takes no arguments; returns 1 when program is not being by'd and is being called with the last by-group; returns 0 otherwise.
function `_byn1()`	takes no arguments; returns the beginning observation number of the by-group currently being executed; returns 1 if `_by()==0`. The value returned by `_byn1()` is valid only if the data have not been re-sorted since the original call to the `by` program.
function `_byn2()`	takes no arguments; returns the ending observation number of the by-group currently being executed; returns 1 if `_by()==0`. The value returned by `_byn2()` is valid only if the data have not been re-sorted since the original call to `by` program.
macro `‘_byindex’`	contains nothing when program is not being by'd; contains name of temporary variable when program is being by'd: variable contains 1, 2, ... for each observation in data and recorded value indicates to which by-group each observation belongs.
macro `‘_byvars’`	contains nothing when program is not being by'd; contains names of the actual by-variables otherwise.
macro `‘_byrc0’`	contains "`, rc0`" if the `rc0` option is specified; contains nothing otherwise.

So let's consider the problems one at a time, beginning with the second problem. Your program does not use `marksample`, and we will assume that your program has good reason for not doing so, because the easy fix would be to use `marksample`. Still, your program must somehow be determining which observations to use, and we will assume that you are creating a `‘touse’` temporary variable containing 0 if the observation is to be omitted from the analysis and 1 if it is to be used. Somewhere, early in your program, you are setting the `‘touse’` variable. Right after that, make the following addition (shown in bold):

```
program ..., ... byable(recall)
        ...
        if _by() {
                quietly replace ‘touse’ = 0 if ‘_byindex’ != _byindex()
        }
        ...
end
```

The fix is easy: you ask if you are being by'd and, if so, you set `‘touse’` to 0 in all observations for which the value of `‘byindex’` is not equal to the by-group you are currently considering, namely, `_byindex()`.

The first problem is also easy to fix. Say that your program has a `generate(`*newvar*`)` option. Your code must therefore contain

```
program ..., ...
        ...
        if "`generate'" != "" {
                ...
        }
        ...
end
```

Change the program to read

```
program ..., ... byable(recall)
        ...
        if "`generate'" != "" & _bylastcall() {
                ...
        }
        ...
end
```

`_bylastcall()` will be 1 (meaning true) whenever your program is not being by'd and, when it is being by'd, whenever the program is being executed for the last by-group. The result is that the new variable will be created containing only the values for the last by-group, but with a few exceptions, that is how all of Stata works. Alternatives are discussed under `byable(onecall)`.

All the other macros and functions that are available are for creating special effects and are rarely used in `byable(recall)` programs.

Using sort in byable(recall) programs

You may use `sort` freely within `byable(recall)` programs, and in fact, you can use any other Stata command you wish; there are simply no issues. You may even use `sortpreserve` to restore the sort order at the conclusion of your program; see [P] **sortpreserve**.

We will discuss the issue of `sort` in depth just to convince you that there is nothing with which you must be concerned.

When a `byable(recall)` program receives control and is being by'd, the data are guaranteed to be sorted by `` `_byvars' `` only when `_byindex()` = 1—only on the first call. If the program re-sorts the data, the data will remain re-sorted on the second and subsequent calls, even if `sortpreserve` is specified. This may sound like a problem, but it is not. `sortpreserve` is not being ignored; the data will be restored to their original order after the final call to your program. Let's go through the two cases: either your program uses `sort` or it does not.

1. If your program needs to use `sort`, it will probably need a different sort order for each by-group. For instance, a typical program that uses `sort` will include lines such as

   ```
   sort `touse' `id' ...
   ```

   and so move the relevant sample to the top of the dataset. This `byable(recall)` program makes no reference to the `` `_byvars' `` themselves, nor does it do anything differently when the `by` prefix is specified and when it is not. That is typical; `byable(recall)` programs rarely find it necessary to refer to the `` `_byvars' `` directly.

   In any case, because this program is sorting the data explicitly every time it is called (and we know it must be because `byable(recall)` programs are executed once for each by-group), there is no reason for Stata to waste its time restoring a sort order that will just be undone anyway. The original sort order needs to be reestablished only after the final call.

2. The other alternative is that the program does not use `sort`. Then it is free to exploit that the data are sorted on `'_byvars'`. Because the data will be sorted on the first call, the program does no `sort`s, so the data will be sorted on the second call, and so on. `byable(recall)` programs rarely exploit the sort order, but the program is free to do so.

Byable estimation commands

Estimation commands are natural candidates for the `byable(recall)` approach. There is, however, one issue that requires special attention. Estimation commands really have two syntaxes: one at the time of estimation,

[*prefix_command*:] *estcmd varlist* ... [, *estimation_options replay_options*]

and another for redisplaying results:

estcmd [, *replay_options*]

With estimation commands, `by` is not allowed when results are redisplayed. We must arrange for this in our program, and that is easy enough. The general outline for an estimation command is

```
program estcmd, ...
        if replay() {
                if "`e(cmd)'"!="estcmd"  error 301
                syntax [, replay_options]
        }
        else {
                syntax ... [, estimation_options  replay_options]
                ... estimation logic...
        }
        ... display logic...
```

and to this, we make the changes shown in bold:

```
program estcmd, ... byable(recall)
        if replay() {
                if "`e(cmd)'"!="estcmd" error 301
                if _by() error 190
                syntax [, replay_options]
        }
        else {
                syntax ... [, estimation_options  replay_options]
                ... estimation logic...
        }
        ... display logic...
```

In addition to adding `byable(recall)`, we add the line

```
                if _by() error 190
```

in the case where we have been asked to redisplay results. If we are being by'd (if `_by()` is true), then we issue error 190 (request may not be combined with by).

byable(onecall) programs

byable(onecall) requires more work to use. We strongly recommend using byable(recall) whenever possible.

The main use of byable(onecall) is to create programs such as generate and egen, which allow the by prefix but operate on all the data and create a new variable containing results for all the different by-groups.

byable(onecall) programs are, as the name implies, executed only once. The byable(onecall) program is responsible for handling all the issues concerning the by, and it is expected to do that by using

function _by()	takes no arguments returns 0 when program is not being by'd returns 1 when program is being by'd
macro '_byvars'	contains nothing when program is not being by'd contains names of the actual by-variables otherwise
macro '_byrc0'	contains nothing or "rc0" contains ", rc0" if by's rc0 option was specified

In byable(onecall) programs, you are responsible for everything, including the output of by-group headers if you want them.

The typical candidates for byable(onecall) are programs that do something special and odd with the by-variables. We offer the following guidelines:

1. Ignore that you are going to make your program byable when you first write it. Instead, include a by() option in your program. Because your program cannot be coded using byable(recall), you already know that the by-variables are entangled with the logic of your routine. Make your program work before worrying about making it byable.
2. Now go back and modify your program. Include byable(onecall) on the program statement line. Remove by(varlist) from your syntax statement, and immediately after the syntax statement, add the line

   ```
   local by "'_byvars'"
   ```

3. Test your program. If it worked before, it will still work now. To use the by() option, you put the by *varlist*: prefix out front.
4. Ignore the macro '_byrc0'. Byable programs rarely do anything different when the user specifies by's rc0 option.

Using sort in byable(onecall) programs

You may use sort freely within byable(onecall) programs. You may even use sortpreserve to restore the sort order at the conclusion of your program.

When a byable(onecall) program receives control and is being by'd, the data are guaranteed to be sorted by '_byvars'.

Combining byable(onecall) with byable(recall)

`byable(onecall)` can be used as an interface to other byable programs. Let's pretend that you are writing a command—we will call it `switcher`—that calls one of two other commands based perhaps on some aspect of what the user typed or, perhaps, based on what was previously estimated. The rule by which `switcher` decides to call one or the other does not matter for this discussion; what is important is that `switcher` switches between what we will call `prog1` and `prog2`. `prog1` and `prog2` might be actual Stata commands, Stata commands that you have written, or even subroutines of `switcher`.

We will further imagine that `prog1` and `prog2` have been implemented using the `byable(recall)` method and that we now want `switcher` to allow the `by` prefix, too. The easy way to do that is

```
program switcher, byable(onecall)
        if _by() {
                local by "by `_byvars' `_byrc0':"
        }
        if (whatever makes us decide in favor of prog1) {
                `by' prog1 `0'
        }
        else    `by' prog2 `0'
end
```

`switcher` works by re-creating the `by` *varlist*`:` prefix in front of `prog1` or `prog2` if `by` was specified. `switcher` will be executed only once, even if `by` was specified. `prog1` and `prog2` will be executed repeatedly.

In the above outline, it is not important that `prog1` and `prog2` were implemented using the `byable(recall)` method. They could just as well be implemented using `byable(onecall)`, and `switcher` would change not at all.

The by-group header

Usually, when you use a command with `by`, a header is produced above each by-group:

```
. by foreign:  summarize mpg weight
-------------------------------------------------------------------------------
-> foreign = Domestic
  (output for first by-group appears)
-------------------------------------------------------------------------------
-> foreign = Foreign
  (output for first by-group appears)
. _
```

The by-group header does not always appear:

```
. by foreign:  generate new = sum(mpg)
. _
```

When you write your own programs, the header will appear by default if you use `byable(recall)` and will not appear if you use `byable(onecall)`.

If you want the header and use `byable(onecall)`, you will have to write the code to output it.

If you do not want the header and use `byable(recall)`, you can specify `byable(recall, noheader)`:

```
program ..., ... byable(recall, noheader)
        ...
end
```

Also see

[P] **program** — Define and manipulate programs

[P] **sortpreserve** — Sort within programs

[D] **by** — Repeat Stata command on subsets of the data