frame prefix — The frame prefix command

Description Quick start Syntax Remarks and examples Also see

Description

The `frame` prefix allows you to execute one or more Stata commands in another frame, leaving the current frame unchanged.

Quick start

Describe the data in frame `fr1`

```
frame fr1: describe
```

Execute a series of commands in frame `fr2`

```
frame fr2 {
  use mydata
  summarize
  codebook
}
```

Syntax

```
frame framename: stata_command
```

```
frame framename {
      commands to execute in context of framename
}
```

Remarks and examples

Remarks are presented under the following headings:

Example of interactive use

You have data in two frames. In your current frame you have data containing detailed information on sales for your company across four regions. A colleague just sent you an email with a summary dataset named `sales.dta`, which is supposed to contain the total sales for each region. You want to make sure the summary dataset was created from the same base sales information as the detailed dataset.

In your current dataset, you know from `summarize` that the total sales for the South region were $532,399 and the total cost of the goods sold was $330,499. You check that the dataset you just received matches these totals:

```
. frame create summary
. frame summary: use sales
. frame summary: list if region=="South"
```

The frame prefix command allowed you to load a dataset in frame summary and run a command on that data without affecting anything in your current frame.

Example of use in programs

The frame prefix can be used for one-liners, such as above, or it can be used to execute a whole series of commands on the data in another frame. The nice thing in either case is that no matter what happens when those commands are executed, whether they complete successfully or exit with error, the current frame will come back to what it was before you called the frame prefix command. In programs, this means that you do not have to hold on to the current frame name and change back to it after working in another frame.

You are writing a program that takes a subset of the current data, performs some manipulations on that subset, and then graphs the result. The required manipulations would damage the original dataset. One way to do this would be to

1. create a temporary frame:

```
tempname tmpframe
```

2. put a subset of data into it:

```
frame put if ..., into(‘tmpframe’)
```

3. perform the needed manipulations and graph the result:

```
frame ‘tmpframe’ {
        some commands which manipulate the data
        graph twoway ...
}
```

At the end of this block of code, any commands that appear next will work against the original frame, not ‘tmpframe’. You could add a line to drop ‘tmpframe’, but there is no need. Because it has a temporary name, the frame and the data in it will automatically be dropped when your program or do-file completes.

An alternative workflow for the above would be to first preserve your data, then manipulate them in place and obtain your graph. You could then restore the original data. Whether you should use the frame prefix approach or the preserve and restore approach is up to you. The frame approach is often faster, but if your dataset in memory is extremely large, you may not want to make another entire copy of it in memory, even temporarily, and thus, the second approach may be better in such a case.

Also see

[D] **frames intro** — Introduction to frames

[D] **frames** — Data frames