st_frame*() — Data frame manipulation

Description   Syntax   Remarks and examples   Conformability   Diagnostics   Also see

Description

st_framecurrent() returns the name of current (working) frame.

st_framedir() returns the names of all existing frames.

st_framecreate() makes a new frame without making it the current frame.

st_framecurrent() changes the identity of current (working) frame.

st_framerename() renames the existing frame, which can be the current frame.

st_framedrop() drops or eliminates the frame that is not the current frame.

st_framedropabc() drops or eliminates all frames except the current frame. abc stands for "all but current".

st_framereset() resets Stata or Mata to contain one empty frame named default.

st_framecopy() copies or duplicates complete contents from one frame to another, clearing the previous contents of the target frame if necessary.

st_frameexists() determines whether the frame named *name* exists.

_st_framecreate(), _st_framecurrent(), _st_framerename(), _st_framedrop(), and _st_framec
perform the same action as st_framecreate(), st_framecurrent(), st_framerename(), st_framedrop
and st_framecopy(), respectively, except that they handle errors differently. The functions without a leading underscore issue an error message, display a traceback log, and abort execution when used incorrectly. The functions with a leading underscore do not abort. They return a nonzero value and execution continues.

For an overview of frames, see [D] **frames intro**.

Syntax

string scalar	`st_framecurrent()`
string colvector	`st_framedir()`
void	`st_framecreate(`*fname*`)`
real scalar	`_st_framecreate(`*fname*`,` *noisy*`)`
void	`st_framecurrent(`*fname*`)`
real scalar	`_st_framecurrent(`*fname*`,` *noisy*`)`
void	`st_framerename(`*fname*`,` *newfname*`)`
real scalar	`_st_framerename(`*fname*`,` *newfname*`,` *noisy*`)`
void	`st_framedrop(`*fname*`)`
real scalar	`_st_framedrop(`*fname*`,` *noisy*`)`
void	`st_framedropabc()`
void	`st_framereset()`
void	`st_framecopy(`*fname_to*`,` *fname_from*`)`
real scalar	`_st_framecopy(`*fname_to*`,` *fname_from*`,` *noisy*`)`
real scalar	`st_frameexists(`*name*`)`

where

fname is a *string scalar* containing a name of an existing frame.

newfname is a *string scalar* containing a name that is not the name of an existing frame.

name is a *string scalar* containing a name, whether or not of an existing frame.

noisy is a *real scalar* containing 0 (error messages suppressed) or any nonzero value (error messages shown).

Remarks and examples

Stata allows more than one dataset to be stored in memory. Each is stored in a separate frame, which you name. For an overview of frames, see [D] **frames intro**. The `st_frame*()` functions let you create new frames, delete existing ones, and switch the identity of the current or working frame from one frame to another. Stata commands and Mata functions work on the current (working) frame. Data from more than one frame may be accessed simultaneously by creating Mata views onto those frames and using them in expressions.

Notice that some of the `st_frame*()` commands are paired:

```
           void st_framecreate(...)
    real scalar _st_framecreate(..., noisy)

           void st_framecurrent(...)
    real scalar _st_framecurrent(..., noisy)

           void st_framerename(...)
    real scalar _st_framerename(..., noisy)

           void st_framedrop(...)
    real scalar _st_framedrop(..., noisy)

           void st_framecopy(...)
    real scalar _st_framecopy(..., noisy)
```

The paired functions do the same thing but handle errors differently. The functions without a leading underscore issue an error message, display a traceback log, and abort execution when used incorrectly. For example,

```
: st_framecreate("default")
frame name default already exists
        st_framecreate():   3598  Stata returned error
                  <istm>:      -  function returned error
r(3598);
```

The functions with a leading underscore do not abort. They return a nonzero value and execution continues. Consider the following function:

```
void example()
{
        rc = _st_framecreate("default", 1)
        printf("execution continues, rc = %f\n", rc)
}
```

Execution of it results in

```
: example()
frame name default already exists
execution continues, rc = 110
```

The error message appeared but execution continued, and the error message appeared only because we coded `1` for *noisy* in the call to `_st_framecreate()`:

```
rc = _st_framecreate("default", 1)
```

Had we coded `0`, the error message would not have appeared, but execution would still have continued, and we would still see the execution-continues message, and `rc` would have still contained 110.

The 110 is an example of a Stata return code. Stata return codes are 0 when the function runs without error. The number 110 is the particular code for already exists. Something already existed, in this case, the frame name. If we had illustrated return codes using `_st_framedrop()` and specified a frame name that did not exist, the return code would have been 111, meaning something does not exist, that something being the frame name.

The underscore variants exist to allow you to write more elegant code in which the output does not suggest to the user that your code has a bug when it was in fact used incorrectly by the user. We could have written `example()` as

```
void example()
{
        rc = _st_framecreate("default", 1)
        if (rc!=0) exit(rc)
        printf("execution continues, rc = %f\n", rc)
}
```

and then the output would have been

```
: example()
frame name default already exists
r(110);
```

Conformability

All arguments to the `st_frame*()` and `_st_frame*()` functions are 1 × 1.

Diagnostics

`st_framecurrent()`, `st_framedir()`, `st_framedropabc()`, and `st_frameexists()` always run successfully. The other `st_frame*()` commands abort execution when errors occur.

The `_st_frame*()` commands never abort. They return 0 or, when errors occur, the relevant nonzero Stata return code.

Also see

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

[M-5] **st_store()** — Modify values stored in current Stata dataset

[M-5] **st_view()** — Make matrix that is a view onto current Stata dataset

[M-4] **Stata** — Stata interface functions

[D] **putmata** — Put Stata variables into Mata and vice versa