

**Description**

`stata(cmd)` executes the Stata command contained in the string scalar `cmd`. Output from the command appears at the terminal, and any macros contained in `cmd` are expanded.

`stata(cmd, nooutput)` does the same thing, but if `nooutput ≠ 0`, output produced by the execution is not displayed. `stata(cmd, 0)` is equivalent to `stata(cmd)`.

`stata(cmd, nooutput, nomacroexpand)` does the same thing but, before execution, suppresses expansion of any macros contained in `cmd` if `nomacroexpand ≠ 0`. `stata(cmd, 0, 0)` is equivalent to `stata(cmd)`.

`_stata()` repeats the syntaxes of `stata()`. The difference is that, whereas `stata()` aborts with error if the execution results in a nonzero return code, `_stata()` returns the resulting return code.

**Syntax**

```
void           stata(cmd)
void           stata(cmd, nooutput)
void           stata(cmd, nooutput, nomacroexpand)

real scalar    _stata(cmd)
real scalar    _stata(cmd, nooutput)
real scalar    _stata(cmd, nooutput, nomacroexpand)
```

where

```
cmd:          string scalar
nooutput:     real scalar
nomacroexpand: real scalar
```

**Remarks and examples**

The command you execute may invoke a process that causes another instance of Mata to be invoked. For instance, Stata program `A` calls Mata function `m1()`, which executes `stata()` to invoke Stata program `B`, which in turn calls Mata function `m2()`, which . . . .

`stata(cmd)` and `_stata(cmd)` execute `cmd` at the current run level. This means that any local macros refer to local macros in the caller’s space. Consider the following:
program example
...
    local x = "value from A"
    mata: myfunc()
    display "'x'"
...
end
mata void myfunc()
{
    stata("local x = "new value"")
}

After example executes mata: myfunc(), 'x' will be "new value".

That _stata() and _ stata() work that way was intentional: Mata functions can modify the caller’s environment so that they may create temporary variables for the caller’s use, etc., and you only have to exercise a little caution. Executing _ stata() functions to run other ado-files and programs will cause no problems because other ado-files and programs create their own new environment in which temporary variables, local macros, etc., are private.

Also, do not use _ stata() or _ stata() to execute a multiline command or to execute the first line of what could be considered a multiline command. Once the first line is executed, Stata will fetch the remaining lines from the caller’s environment. For instance, consider

```
begin myfile.do

mata void myfunc()
{
    stata("if (1==1) {")
    mata: myfunc()
    display "hello"
}
end myfile.do
```

In the example above, myfunc() will consume the display "hello" and } lines.

**Conformability**

```
stata(cmd, nooutput, nomacroexpand):
    cmd: 1 × 1
    nooutput: 1 × 1 (optional)
    nomacroexpand: 1 × 1 (optional)
    result: void

_stata(cmd, nooutput, nomacroexpand):
    cmd: 1 × 1
    nooutput: 1 × 1 (optional)
    nomacroexpand: 1 × 1 (optional)
    result: 1 × 1
```
Diagnostics

\texttt{stata()} aborts with error if \textit{cmd} is too long (exceedingly unlikely), if macro expansion fails, or if execution results in a nonzero return code.

\texttt{\_stata()} aborts with error if \textit{cmd} is too long.

Also see

\texttt{\[M-3\] mata stata} — Execute Stata command

\texttt{\[M-4\] Stata} — Stata interface functions