

`_return` — Preserve stored results

[Description](#)[Syntax](#)[Option](#)[Remarks and examples](#)[Stored results](#)[Also see](#)

## Description

`_return` sets aside and restores the contents of `r()`.

`_return hold` stores under *name* the contents of `r()` and clears `r()`. If *name* is a name obtained from *tempname*, *name* will be dropped automatically at the program's conclusion, if it is not automatically or explicitly dropped before that.

`_return restore` restores from *name* the contents of `r()` and, unless option `hold` is specified, drops *name*.

`_return drop` removes from memory (drops) *name* or, if `_all` is specified, all `_return` names currently saved.

`_return dir` lists the names currently set aside by `_return`.

## Syntax

*Set aside contents of r()*

```
_return hold name
```

*Restore contents of r() from name*

```
_return restore name [ , hold ]
```

*Drop specified \_return name*

```
_return drop {name | _all}
```

*List names currently stored by \_return*

```
_return dir
```

## Option

`hold`, specified with `_return restore`, specifies that results continue to be held so that they can be `_return` restored later, as well. If the option is not specified, the specified results are restored and *name* is dropped.

## Remarks and examples

`_return` is rarely necessary. Most programs open with

```

program example
  version 15.1
  syntax ...
  marksample touse
  if "'exp'" != "" {
    touse e
    qui generate double `e' = `exp' if `touse'
  }
  ... (code to calculate final results)...
end

```

In the program above, no commands are given that change the contents of `r()` until all parsing is complete and the `if exp` and `=exp` are evaluated. Thus the user can type

```

. summarize myvar
. example ... if myvar>r(mean) ...

```

and the results will be as the user expects.

Some programs, however, have nonstandard and complicated syntax, and in the process of deciphering that syntax, other `r`-class commands might be run before the user-specified expressions are evaluated. Consider a command that reads

```

program example2
  version 15.1
  ... (commands that parse)...
  ... (r() might be reset at this stage)...
  ... commands that evaluate user-specified expressions...
  tempvar touse
  mark `touse' `if'
  tempvar v1 v2
  generate double `v1' = `exp1' if `touse'
                                // `exp1' specified by user
  generate double `v2' = `exp2' if `touse'
                                // `exp2' specified by user
  ... (code to calculate final results)...
end

```

Here it would be a disaster if the user typed

```

. summarize myvar
. example2 ... if myvar>r(mean) ...

```

because `r(mean)` would not mean what the user expected it to mean, which is the mean of `myvar`. The solution to this problem is to code the following:

```
program example2
  version 15.1

                                // hold on to r()

  tempname myr
  _return hold 'myr'
  ... (commands that parse)...
  ... (r() might be reset at this stage)...
  ... commands that evaluate user-specified expressions...
                                // restore r()

  _return restore 'myr'

  tempvar touse
  mark 'touse' 'if'
  tempvar v1 v2
  generate double 'v1' = 'exp1' if 'touse'
                                // 'exp1' specified by user
  generate double 'v2' = 'exp2' if 'touse'
                                // 'exp2' specified by user
  ... (code to calculate final results)...

end
```

In the above example, we hold on to the contents of `r()` in `'myr'` and then later bring them back.

## Stored results

`_return restore` restores in `r()` those results that were stored in `r()` when `_return hold` was executed.

## Also see

[P] [return](#) — Return stored results