Title

return — Return stored results

Description Syntax Options Remarks and examples Also see

Description

Results of calculations are stored by many Stata commands so that they can be easily accessed and substituted into subsequent commands. This entry summarizes for programmers how to store results. If your interest is in using previously stored results, see [R] **stored results**.

`return` stores results in `r()`.

`ereturn` stores results in `e()`.

`sreturn` stores results in `s()`.

Stata also has the values of system parameters and certain constants such as pi stored in `c()`. Because these values may be referred to but not assigned, the c-class is discussed in a different entry; see [P] **creturn**.

Syntax

Return results for general commands, stored in r()

`return list` [`, all`]

`return clear`

`return scalar` *name* `=` *exp*

`return local` *name* `=` *exp*

`return local` *name* [`"`]*string*[`"`]

`return matrix` *name* [`=`] *matname* [`, copy`]

`return add`

Return results for estimation commands, stored in e()

```
ereturn list [, all]

ereturn clear

ereturn post [b [V [Cns]]] [weight] [, depname(string) obs(#) dof(#)
  esample(varname) properties(string)]

ereturn scalar name = exp

ereturn local name = exp

ereturn local name ["]string["]

ereturn matrix name [=] matname [, copy]

ereturn repost [b = b] [V = V] [Cns = Cns] [weight] [, esample(varname)
  properties(string) rename]
```

Return results for parsing commands, stored in s()

```
sreturn list

sreturn clear

sreturn local name = exp

sreturn local name ["]string["]
```

where **b**, **V**, and **Cns** are *matnames*, which is the name of an existing matrix.

`fweights`, `aweights`, `iweights`, and `pweights` are allowed; see **[U] 11.1.6 weight**.

Options

`all` is for use with `return list` and `ereturn list`. `all` specifies that hidden and historical stored results be listed along with the usual stored results. This option is seldom used. See *Using hidden and historical stored results* and *Programming hidden and historical stored results* in *Remarks and examples* for more information. These sections are written in terms of `return list`, but everything said there applies equally to `ereturn list`.

`all` is not allowed with `sreturn list` because `s()` does not allow hidden or historical results.

`copy` specified with `return matrix` or `ereturn matrix` indicates that the matrix is to be copied; that is, the original matrix should be left in place. The default is to "steal" or "rename" the existing matrix, which is fast and conserves memory.

`depname(`*string*`)` is for use with `ereturn post`. It supplies the name of the dependent variable to appear in the estimation output. The name specified need not be the name of an existing variable.

`obs(#)` is for use with `ereturn post`. It specifies the number of observations on which the estimation was performed. This number is stored in `e(N)`, and `obs()` is provided simply for convenience. Results are no different from those for `ereturn post` followed by `ereturn scalar N = #`.

`dof(#)` is for use with `ereturn post`. It specifies the number of denominator degrees of freedom to be used with t and F statistics and so is used in calculating significance levels and confidence intervals. The number specified is stored in `e(df_r)`, and `dof()` is provided simply for convenience. Results are no different from those for `ereturn post` followed by `ereturn scalar df_r = #`.

`esample(`*varname*`)` is for use with `ereturn post` and `ereturn repost`. It specifies the name of a 0/1 variable that is to become the `e(sample)` function. *varname* must contain 0 and 1 values only, with 1 indicating that the observation is in the estimation subsample. `ereturn post` and `ereturn repost` will be able to execute a little more quickly if *varname* is stored as a `byte` variable.

varname is dropped from the dataset, or more correctly, it is stolen and stashed in a secret place.

`properties(`*string*`)` specified with `ereturn post` or `ereturn repost` sets the `e(properties)` macro. By default, `e(properties)` is set to `b V` if `properties()` is not specified.

`rename` is for use with the `b = b` syntax of `ereturn repost`. All numeric estimation results remain unchanged, but the labels of `b` are substituted for the variable and equation names of the already posted results.

Remarks and examples

stata.com

Remarks are presented under the following headings:

Introduction
Storing results in r()
Storing results in e()
Storing results in s()
Recommended names for stored results
Using hidden and historical stored results
Programming hidden and historical stored results

Introduction

This entry summarizes information that is presented in greater detail in other parts of the Stata documentation. Most particularly, we recommend that you read [U] **18 Programming Stata**. The commands listed above are used by programmers to store results, which are accessed by others using `r()`, `e()`, and `s()`; see [R] **stored results**.

The commands listed above may be used only in programs—see [U] **18 Programming Stata** and [P] **program**—and then only when the program is declared explicitly as being `rclass`, `eclass`, or `sclass`:

```
program ..., rclass
        ...
        return ...
        ...
end

program ..., eclass
        ...
        ereturn ...
        ...
end
```

```
program ..., sclass
        ...
        sreturn ...
        ...
end
```

Storing results in r()

- The program must be declared explicitly to be r-class: `program ..., rclass`.
- Distinguish between `r()` (returned results) and `return()` (results being assembled that will be returned). The program you write actually stores results in `return()`. Then when your program completes, whatever is in `return()` is copied to `r()`. Thus the program you write can consume `r()` results from other programs, and there is no conflict.
- `return clear` clears the `return()` class. This command is seldom used because `return()` starts out empty when your program begins. `return clear` is for those instances when you have started assembling results and all is going well, but given the problem at hand, you need to start all over again.
- `return scalar` *name* `=` *exp* evaluates *exp* and stores the result in the scalar `return(`*name*`)`. *exp* must evaluate to a numeric result or missing. If your code has previously stored something in `return(`*name*`)`, whether a scalar, matrix, or whatever else, the previous value is discarded and this result replaces it.
- `return local` *name* `=` *exp* evaluates *exp* and stores the result in the macro `return(`*name*`)`. *exp* may evaluate to a numeric or string result. If your code has previously stored something in `return(`*name*`)`, whether a scalar, matrix, or whatever else, the previous value is discarded and this result replaces it.

  Be careful with this syntax: do not code

  `return local` *name* `=` `‘`*mymacro*`’`

  because that will copy just the first 2045 characters of `‘`*mymacro*`’`. Instead, code

  `return local` *name* `‘""‘`*mymacro*`’""’`

- `return local` *name string* copies *string* to macro `return(`*name*`)`. If your code has previously stored something in `return(`*name*`)`, whether a scalar, matrix, or whatever else, the previous value is discarded and this result replaces it.

  If you do not enclose *string* in double quotes, multiple blanks in *string* are compressed into single blanks.
- `return matrix` *name matname* destructively copies `matname` into matrix `return(`*name*`)`, meaning that *matname* is erased (*matname* is renamed `return(`*name*`)`). If your code has previously stored something in `return(`*name*`)`, whether a scalar, matrix, or whatever else, the previous value is discarded and this result replaces it.
- `return add` copies everything new in `r()` into `return()`. Say that your program performed a `summarize`. `return add` lets you add everything just returned by `summarize` to the to-be-returned results of your program. If your program had already set `return(N)`, `summarize`'s `r(N)` would not replace the previously set result. The remaining `r()` results set by `summarize` would be copied.

Storing results in e()

For detailed guidance on storing in `e()`, see [P] **ereturn**. What follows is a summary.

- The program must be declared explicitly to be e-class: `program ..., eclass`.
- The e-class is cleared whenever an `ereturn post` is executed. The e-class is a static, single-level class, meaning that results are posted to the class the instant that they are stored.
- `ereturn clear` clears `e()`. This is a rarely used command.
- `ereturn post` is how you must begin storing results in `e()`. Because `ereturn post` clears `e()`, anything stored in `e()` prior to the `ereturn post` is lost.

  `ereturn post` stores matrix (vector, really) `e(b)`, matrices `e(V)` and `e(Cns)`, weight-related macros `e(wtype)` and `e(wexp)`, and function `e(sample)`. The most common syntax is

  ```
  ereturn post `b' `V', esample(`touse') ...
  ```

  where `` `b' `` is a row vector containing the parameter estimates, `` `V' `` is a symmetric matrix containing the variance estimates, and `` `touse' `` is a 0/1 variable recording 1 in observations that appear in the estimation subsample.

  The result of this command will be that `` `b' ``, `` `V' ``, and `` `touse' `` all disappear. In fact, `ereturn post` examines what you specify and, if it is satisfied with them, renames them `e(b)`, `e(V)`, and `e(sample)`.

  For more advanced usage that also posts constraint and weight information, see [P] **ereturn**.

  In terms of `ereturn post`'s other options,

  a. We recommend that you specify `depname(`*string*`)` if there is one dependent variable name that you want to appear on the output. Whether you specify `depname()` or not, remember later to define macro `e(depvar)` to contain the names of the dependent variables.

  b. Specify `obs(#)`, or remember later to define scalar `e(N)` to contain the number of observations.

  c. Few models require specifying `dof(#)`, or, if that is not done, remembering to later define scalar `e(df_r)`. This all has to do with substituting t and F statistics on the basis of # (denominator) degrees of freedom for asymptotic z and χ^2 statistics in the estimation output.

- `ereturn scalar` *name* `=` *exp* evaluates *exp* and stores the result in the scalar `e(`*name*`)`. *exp* must evaluate to a numeric result or missing. If your code has previously stored something in `e(`*name*`)`, whether that be a scalar, matrix, or whatever else, the previous value is discarded and this result replaces it.
- `ereturn local` *name* `=` *exp* evaluates *exp* and stores the result in the macro `e(`*name*`)`. *exp* may evaluate to a numeric or string result. If your code has previously stored something in `e(`*name*`)`, whether that be a scalar, matrix, or whatever else, the previous value is discarded and this result replaces it.

  Be careful with this syntax: do not code

  ```
  ereturn local name = `mymacro'
  ```

  because that will copy just the first 2045 characters of `` `mymacro' ``. Instead, code

  ```
  ereturn local name `"`mymacro'"'
  ```

- `ereturn local` *name string* copies *string* to macro `e(`*name*`)`. If your code has previously stored something in `e(`*name*`)`, whether a scalar, matrix, or whatever else, the previous value is discarded and this result replaces it.

If you do not enclose *string* in double quotes, multiple blanks in *string* are compressed into single blanks.

- `ereturn matrix` *name* = *matname* destructively copies `matname` into matrix `e(`*name*`)`, meaning that *matname* is erased. At least, that is what happens if you do not specify the `copy` option. What actually occurs is that *matname* is renamed `e(`*name*`)`. If your code has previously stored something in `e(`*name*`)`, whether a scalar, matrix, or whatever else, the previous value is discarded and this result replaces it, with two exceptions:

  `ereturn matrix` cannot be used to store in `e(b)` or `e(V)`. The only way to post matrices to these special names is to use `ereturn post` and `ereturn repost` so that various tests can be run on them before they are made official. Other Stata commands use `e(b)` and `e(V)` and expect to see a valid estimation result. If `e(b)` is $1 \times k$, they expect `e(V)` to be $k \times k$. They expect that the names of rows and columns will be the same so that the ith column of `e(b)` corresponds to the ith row and column of `e(V)`. They expect `e(V)` to be symmetric. They expect `e(V)` to have positive or zero elements along its diagonal, and so on. `ereturn post` and `ereturn repost` check these assumptions.

- `ereturn repost` allows changing `e(b)`, `e(V)`, `e(Cns)`, `e(wtype)`, `e(wexp)`, `e(properties)`, and `e(sample)` without clearing the estimation results and starting all over again. As with `ereturn post`, specified matrices and variables disappear after reposting because they are renamed `e(b)`, `e(V)`, `e(Cns)`, or `e(sample)` as appropriate.

- Programmers posting estimation results should remember to store
  a. Macro `e(cmd)`, containing the name of the estimation command. Make this the last thing you store in `e()`.
  b. Macro `e(cmdline)`, containing the command the user typed.
  c. Macro `e(depvar)`, containing the names of the dependent variables.
  d. Scalar `e(N)`, containing the number of observations.
  e. Scalar `e(df_m)`, containing the model degrees of freedom.
  f. Scalar `e(df_r)`, containing the denominator degrees of freedom if estimates are nonasymptotic; otherwise, do not define this result.
  g. Scalar `e(ll)`, containing the log-likelihood value, if relevant.
  h. Scalar `e(ll_0)`, containing the log-likelihood value for the constant-only model, if relevant.
  i. Scalar `e(chi2)`, containing the χ^2 test of the model against the constant-only model, if relevant.
  j. Macro `e(chi2type)`, containing `LR`, `Wald`, or other, depending on how `e(chi2)` was obtained.
  k. Scalar `e(r2)`, containing the value of the R^2 if it is calculated.
  l. Scalar `e(r2_p)`, containing the value of the pseudo-R^2 if it is calculated.
  m. Macro `e(vce)`, containing the name of the *vcetype* that was specified in the `vce()` option; see [R] ***vce_option***.
  n. Macro `e(vcetype)`, containing the text to appear above standard errors in estimation output, typically `Robust`, or it is undefined.
  o. Macro `e(clustvar)`, containing the name of the cluster variable, if any.
  p. Scalar `e(N_clust)`, containing the number of clusters.
  q. Scalar `e(rank)`, containing the rank of `e(V)`.
  r. Macro `e(predict)`, containing the name of the command that `predict` is to use; if this is blank, `predict` uses the default `_predict`.

s. Macro `e(estat_cmd)`, containing the name of an `estat` handler program if you wish to customize the behavior of `estat`.

t. Macro `e(properties)`, containing properties of the estimation command, typically `b V`, indicating that the command produces a legitimate coefficient vector and VCE matrix.

Storing results in s()

- The program must be declared explicitly to be s-class: `program ..., sclass`.
- The s-class is not cleared automatically. It is a static, single-level class. Results are posted to `s()` the instant they are stored.
- `sreturn clear` clears `s()`. We recommend that you use this command near the top of s-class routines. `sreturn clear` may be used in non–s-class programs, too.
- The s-class provides macros only and is intended for returning results of subroutines that parse input. At the parsing step, it is important that the r-class not be changed or cleared because some of what still awaits being parsed might refer to `r()`, and the user expects those results to substitute according to what was in `r()` when he or she typed the command.
- `sreturn local` *name* `=` *exp* evaluates *exp* and stores the result in the macro `s(`*name*`)`. *exp* may evaluate to a numeric or string result. If your code has previously stored something else in `s(`*name*`)`, the previous value is discarded and this result replaces it.

  Be careful with this syntax: do not code

  ```
  sreturn local name = `mymacro'
  ```

  because that will copy just the first 2045 characters of `` `mymacro' ``. Instead, code

  ```
  sreturn local name `"`mymacro'"'
  ```

- `sreturn local` *name string* copies *string* to macro `s(`*name*`)`. If your code has previously stored something else in `s(`*name*`)`, the previous value is discarded and this result replaces it.

  If you do not enclose *string* in double quotes, multiple blanks in *string* are compressed into single blanks.

Recommended names for stored results

Users will appreciate it if you use predictable names for your stored results. We use these rules:

- Mathematical and statistical concepts such as number of observations and degrees of freedom are given short mathematical-style names. Subscripting is indicated with '_'. Names are to proceed from the general to the specific. If `N` means number of observations, `N_1` might be the number of observations in the first group.

  Suffixes are to be avoided where possible. For instance, a χ^2 statistic would be recorded in a variable starting with `chi2`. If, in the context of the command, a statement about "the χ^2 statistic" would be understood as referring to this statistic, then the name would be `chi2`. If it required further modification, such as χ^2 for the comparison test, then the name might be `chi2_c`.

Common prefixes are

N	number of observations
df	degrees of freedom
k	count of parameters
n	generic count
lb and ub	lower and upper bound of confidence interval
chi2	χ^2 statistic
t	t statistic
F	F statistic
p	significance
p and pr	probability
ll	log likelihood
D	deviance
r2	R^2

- Programming concepts, such as lists of variable names, are given English-style names. Names should proceed from the specific to the general. The name of the dependent variable is depvar, not vardep.

  Some examples are

depvar	dependent variable names
eqnames	equation names
model	name of model fit
xvar	x variable
title	title used

- Popular usage takes precedence over the rules. For example:
  a. mss is model sum of squares, even though, per the first rule of this section, it ought to be ss_m.
  b. mean is used as the prefix to record means.
  c. Var is used as the prefix to mean variance.
  d. The returned results from most Stata commands follow this rule.

Using hidden and historical stored results

Most results stored in r() and e() are visible—type return list. Sometimes, other stored results exist, too. For instance, consider the Stata command summarize. Let's pretend that in addition to everything that summarize stores in r()—you know about r(N), r(mean), r(sd), etc.—summarize also stores r(secret) and r(sigma). summarize does not do this, but pretend that it did. If summarize stored r(secret) as hidden and r(sigma) as historical, you would not know they existed from the output of return list unless you typed return list, all. If you typed that command, you would discover r(secret) and r(sigma), and you might learn from the output that r(secret) was hidden whereas r(sigma) was historical. The output is trying to tell you 1) the two stored results exist, 2) you may use them just as you use any other stored result, and 3) the reason why the two stored results were not listed by default.

There are two reasons why summarize might not store results so that you can see them when you type return list.

The first reason is that summarize is designed to work tightly with some other Stata subroutine and is using r() to pass complicated information. The information that is stored is so arcane that you would not want to read documentation about it. Stata puts such stored results into the hidden category where you will not see them by default. If you type return list, all and find hidden

stored results, we recommend that you do not use their contents in your own do- and ado-files. Because hidden stored results are not documented, their names, contents, and even their existence could change in future releases.

The other reason `summarize` might omit a stored result from `return list` concerns backward compatibility. Assume that for Stata 4, `summarize` stored the standard deviation in `r(sigma)` instead of `r(sd)`. Assume that the editors at StataCorp decided later that `r(sd)` would be a better name. The programmers at StataCorp could not simply change the name from `r(sigma)` to `r(sd)`, because users might have already written do- or ado-files before the change. Changing the name could break old do- and ado-files, and it is a hallmark of Stata that your code will continue to work regardless of how long ago users wrote it. Thus the programmers at StataCorp could choose to store the standard deviation in both `r(sigma)` and `r(sd)` in all cases, or they could store the standard deviation in `r(sd)` and store it in `r(sigma)` only when the old do- or ado-file explicitly included a `version 4` or earlier statement. Either way, `r(sigma)` is of no interest to modern Stata users, and so the programmers mark `r(sigma)` as historical. Now when you type `return list`, you will not see `r(sigma)` mentioned; and when you type `return list, all`, you will see `r(sigma)` listed, and you are told that it was not mentioned earlier because it is marked as historical.

Typing `return list, all` can be useful when you are debugging or adding new features to an old program and want to see the historical stored results to better understand your old program.

What was just said about `r()` and `return list` applies equally to `e()` and `ereturn list`, and it applies equally to user-written additions to Stata and to official Stata commands. That's the story of `all`.

Programmers wishing to exploit the hidden and historical markings in their own programs should see the next section.

Programming hidden and historical stored results

You can mark stored results as hidden or historical by specifying the optional *hcat* argument with the appropriate `return` or `ereturn` command:

```
return [hcat] scalar name = exp
return [hcat] local name = exp
return [hcat] local name [ " ]string[ " ]
return [hcat] matrix name [=] matname [ , copy]

ereturn [hcat] scalar name = exp
ereturn [hcat] local name = exp
ereturn [hcat] local name [ " ]string[ " ]
ereturn [hcat] matrix name [=] matname [ , copy]
```

hcat specifies the hiddenness of the result and may be

```
visible
hidden
historical[ (relno) ]
```

where *relno* is #[#][.[#[#]]] such as 2, 10, 10., 10.1, or 10.12. `visible` is the default when *hcat* is not specified.

Thus if you are writing an r-class command and wish to store `r(private)` as a hidden scalar, you can code

```
return hidden scalar private = ...
```

If you wish to store `r(lastvar)` as a hidden local, you can code

```
return hidden local lastvar "..."
```

If you wanted `r(lastvar)` to be historical rather than hidden, you would code

```
return historical local lastvar "..."
```

If you wanted `r(lastvar)` to be historical as of Stata 14, meaning that `r(lastvar)` was current up to but not including Stata 14, you would code

```
return historical(14) local lastvar "..."
```

If you wish to create `r(X)` as a hidden matrix, you can code

```
return hidden matrix X = ...
```

All the above examples could be performed using `ereturn` instead of `return`. They could not be performed using `sreturn` because `s()` does not allow hidden or historical results.

The Mata commands for setting `r()` and `e()` also allow an optional argument to set *hcat*; see [M-5] **st_numscalar()**, [M-5] **st_global()**, and [M-5] **st_matrix()**.

Also see

[P] **creturn** — Return c-class values

[P] **ereturn** — Post the estimation results

[P] **_estimates** — Manage estimation results

[P] **putexcel** — Export results to an Excel file

[P] **_return** — Preserve stored results

[R] **stored results** — Stored results

[U] **18 Programming Stata**

[U] **18.10 Storing results**