program properties — Properties of user-defined programs

Description Option Remarks and examples Also see

Description

User-defined programs can have properties associated with them. Some of Stata's prefix commands—such as `svy` and `stepwise`—use these properties for command validation. You can associate program properties with programs by using the `properties()` option of `program`.

```
program [define] command [ , properties(namelist) ... ]
    // body of the program
end
```

You can retrieve program properties of *command* by using the `properties` macro function.

```
global mname : properties command
```

```
local lclname : properties command
```

Option

`properties(`*namelist*`)` states that *command* has the specified properties. *namelist* may contain up to 80 characters, including separating spaces.

Remarks and examples

Remarks are presented under the following headings:

Introduction

Properties provide a way for a program to indicate to other programs that certain features have been implemented. Suppose that you want to use `stepwise` with the `lr` option so that likelihood-ratio tests are performed in the model-selection process; see [R] **stepwise**. To do that, `stepwise` must know that the estimation command you are using in conjunction with it is a maximum likelihood estimator. If a command declares itself to have the `swml` property, `stepwise` knows that the command can be used with likelihood-ratio tests.

The next few sections discuss properties that are checked by some of Stata's prefix commands and how to make your own programs work with those prefix commands.

Writing programs for use with nestreg and stepwise

Some of Stata's estimation commands can be used with the `nestreg` and `stepwise` prefix commands; see [R] **nestreg** and [R] **stepwise**. For example, the syntax diagram for the `regress` command could be presented as

```
[ nestreg, ...: ] regress ...
```

or

```
[ stepwise, ...: ] regress ...
```

In general, the syntax for these prefix commands is

prefix_command [, *prefix_options*] : *command depvar* (*varlist*) [(*varlist*) ...]
[*if*] [*in*] [, *options*]

where *prefix_command* is either `nestreg` or `stepwise`.

You must follow some additional programming requirements to write programs (ado-files) that can be used with the `nestreg` and `stepwise` prefix commands. Some theoretical requirements must be satisfied to justify using `nestreg` or `stepwise` with a given command.

- *command* must be `eclass` and accept the standard estimation syntax; see [P] **program**, [P] **syntax**, and [P] **mark**.

  command varlist [*if*] [*in*] [*weight*] [, *options*]

- *command* must store the model coefficients and ancillary parameters in `e(b)` and the estimation sample size in `e(N)`, and it must identify the estimation subsample in `e(sample)`; see [P] **ereturn**.

- For the likelihood-ratio test, *command* must have property `swml`. For example, the program definition for `poisson` appears as

  ```
  program poisson, ... properties(... swml ...)
  ```

  command must also store the log-likelihood value in `e(ll)` and the model degrees of freedom in `e(df_m)`.

- For the Wald test, *command* must have property `sw` if it does not already have property `swml`. For example, the program definition for `qreg` appears as

  ```
  program qreg, ... properties(... sw ...)
  ```

  command must also store the variance estimates for the coefficients and ancillary parameters in `e(V)`; see [R] **test**.

Writing programs for use with svy

Some of Stata's estimation commands can be used with the `svy` prefix; see [SVY] **svy**. For example, the syntax diagram for the `regress` command could be presented as

```
[ svy, ...: ] regress ...
```

In general, the syntax for the `svy` prefix is

`svy` [, *svy_options*] : *command varlist* [*if*] [*in*] [, *options*]

You must follow some additional programming requirements to write programs (ado-files) that can be used with the svy prefix. The extra requirements imposed by the svy prefix command are from the various variance-estimation methods that it uses: vce(bootstrap), vce(brr), vce(jackknife), vce(sdr), and vce(linearized). Each of these variance-estimation methods has theoretical requirements that must be satisfied to justify using them with a given command.

- *command* must be eclass and allow iweights and accept the standard estimation syntax; see [P] **program**, [P] **syntax**, and [P] **mark**.

  command varlist [*if*] [*in*] [*weight*] [, *options*]

- *command* must store the model coefficients and ancillary parameters in e(b) and the estimation sample size in e(N), and it must identify the estimation subsample in e(sample); see [P] **ereturn**.

- svy's vce(bootstrap), vce(brr), and vce(sdr) require that *command* have svyb as a property. For example, the program definition for regress appears as

  ```
  program regress, ... properties(... svyb ...)
  ```

- vce(jackknife) requires that *command* have svyj as a property.

- vce(linearized) has the following requirements:

  a. *command* must have svyr as a property.

  b. predict after *command* must be able to generate scores with the following syntax:

     predict [*type*] *stub** [*if*] [*in*], scores

     This syntax implies that estimation results with k equations will cause predict to generate k new equation-level score variables. These new equation-level score variables are *stub*1 for the first equation, *stub*2 for the second equation, ..., and *stubk* for the last equation. Actually svy does not strictly require that these new variables be named this way, but this is a good convention to follow.

     The equation-level score variables generated by predict must be of the form that can be used to estimate the variance by using Taylor linearization (otherwise known as the delta method); see [SVY] **Variance estimation**.

  c. *command* must store the model-based variance estimator for the coefficients and ancillary parameters in e(V); see [SVY] **Variance estimation**.

Writing programs for use with mi

Stata's mi suite of commands provides multiple imputation to provide better estimates of parameters and their standard errors in the presence of missing values; see [MI] **Intro**. Estimation commands intended for use with the mi estimate prefix (see [MI] **mi estimate**) must have property mi, indicating that the command meets the following requirements:

- The command is eclass.
- The command stores its name in e(cmd).
- The command stores the model coefficients and ancillary parameters in e(b), stores the corresponding variance matrix in e(V), stores the estimation sample size in e(N), and identifies the estimation subsample in e(sample).

- The command stores the number of ancillary parameters in `e(k_aux)`. This information is used for the model F test, which is reported by `mi estimate` when the command stores model degrees of freedom in `e(df_m)`.
- If the command employs a small-sample adjustment for tests of coefficients and reports of confidence intervals, the command stores the numerator (residual) degrees of freedom in `e(df_r)`.
- Because `mi estimate` uses its own routines to display the output, to ensure that results display well the command also stores its title in `e(title)`. `mi estimate` also uses macros `e(vcetype)` or `e(vce)` to label the within-imputation variance, but those macros are usually set automatically by other Stata routines.

Properties for survival-analysis commands

Stata's `st` suite of commands have the `st` program property, indicating that they have the following characteristics:

- The command should only be run on data that have been previously `stset`; see [ST] **stset**.
- No dependent variable is specified when calling that command. All variables in *varlist* are regressors. The "dependent" variable is time of failure, handled by `stset`.
- Weights are not specified with the command but instead obtained from `stset`.
- If robust or replication-based standard errors are requested, the default level of clustering is according to the ID variable that was `stset`, if any.

Properties for prefix commands

In addition to checking commands for certain properties, Stata's prefix commands have their own properties. For example, all prefix commands have the `prefix` property; additionally, all commands that have two-part names have the `twopart` property.

For example, the program definition for `bootstrap` looks something like the following:

```
program bootstrap, ... properties(... prefix ...)
```

On the other hand, the program definition for `mi estimate` looks something like the following:

```
program mi, ... properties(... prefix twopart ...)
```

`estimate` is one of many subcommands for `mi`.

Properties for disabling collection of results

Stata's collection system allows you to collect results from many Stata commands and create customized tables with those results. Results can be collected using the `collect` prefix or the `command()` option with `table`. However, not all Stata commands store results, and those commands that do store results do not necessarily make them available for the collection system. The `nocollect` property indicates that the command is not supported by the collection system.

Properties for exponentiating coefficients

Stata has several prefix commands—such as `bootstrap`, `jackknife`, and `svy`—that use alternative variance-estimation techniques for existing commands. These prefix commands behave like conventional estimation commands when reporting and saving estimation results. Given the appropriate program properties, these prefix commands can also report exponentiated coefficients. In fact, the property names for the various shortcuts for the `eform()` option are the same as the option names:

option/*property*	Description
`hr`	hazard ratio
`nohr`	coefficient instead of hazard ratio
`shr`	subhazard ratio
`noshr`	coefficient instead of subhazard ratio
`irr`	incidence-rate ratio
`or`	odds ratio
`rrr`	relative-risk ratio

For example, the program definition for `logit` looks something like the following:

```
program logit, ... properties(... or ...)
```

Putting it all together

`logit` can report odds ratios, works with `svy`, and works with `stepwise`. The program definition for `logit` reads

```
program logit, ... properties(or svyb svyj svyr swml mi) ...
```

Checking for program properties

You can use the `properties` macro function to check the properties associated with a program; see [P] **macro**. For example, the following macro retrieves and displays the program properties for `logit`.

```
. local logitprops : properties logit
. display "‘logitprops’"
or svyb svyj svyr swml mi bayes
```

Also see

[P] **program** — Define and manipulate programs

[MI] **mi estimate** — Estimation using multiple imputations

[R] **nestreg** — Nested model statistics

[R] **stepwise** — Stepwise estimation

[SVY] **svy** — The survey prefix command

[U] 20 Estimation and postestimation commands

For suggested citations, see the FAQ on citing Stata documentation.