

Title

intro — Introduction to programming manual

Description

This entry describes this manual and what has changed since Stata 8.

Remarks

In this manual, you will find

1. matrix-manipulation commands, which are available from the Stata command line and for ado-programming; for advanced matrix functions and a complete matrix programming language, see the [M] *Mata Reference Manual*.
2. commands for programming Stata, and
3. commands and discussions of interest to programmers.

This manual is referred to as [P] in cross-references and is organized alphabetically.

If you are new to Stata's programming commands, we recommend that you first read the chapter about programming Stata in the *User's Guide*; see [U] **18 Programming Stata**. After you read this chapter, we recommend that you read the following sections from this manual:

[P] program	Define and manipulate programs
[P] sortpreserve	Sorting within programs
[P] byable	Making programs byable
[P] macro	Macro definition and manipulation

You may also find the subject table of contents, which immediately follows the table of contents, helpful.

We also recommend the Stata NetCoursesTM. At the time that this introduction was written, our current offerings of Stata programming NetCourses included

- NC-151 An introduction to Stata programming
- NC-152 Advanced Stata programming

To learn more about NetCourses and to view the current offerings of NetCourses, visit <http://www.stata.com/netcourse>.

To learn about writing your own maximum-likelihood estimation commands, read the book *Maximum-likelihood Estimation with Stata*.

What's new

1. The most important change in Stata programming is not found in [P], but rather in [M]—the *Mata Reference Manual*. Mata is an integrated, matrix programming language that is byte compiled, making it very fast. Mata looks much like C, with the extension that all expressions support real and complex matrices. Mata can be used as a matrix calculator, for developing matrix functions and solutions, and for full-fledged program development. Mata has direct access to Stata data, results, and macros; and Mata functions can be called directly from Stata. For complete information, see [M] *Mata Reference Manual*.

2. The new command `viewsource` searches for the specified file along the `adopath` and displays the file in the Viewer. This works not only for ado programs, but also for Mata functions that are programmed themselves in Mata; see [P] [viewsource](#).
3. Programmers of estimation commands or commands that work with estimation results can tie postestimation analysis facilities into the new `estat` (see [R] [estat](#)), making their postestimation facilities behave just like those shipped with Stata; see [P] [estat programming](#).
4. The new `matlist` command provides extensive format control for displaying a matrix; see [P] [matlist](#).
5. The new command `matrix dissimilarity` computes similarity, dissimilarity, or distance matrices from a varlist; see [P] [matrix dissimilarity](#). The `cluster` command (see [MV] [cluster](#)) can now work directly on dissimilarity matrices, and the new `mdsmat` command (see [MV] [mdsmat](#)) can perform multidimensional scaling from a dissimilarity matrix.
6. Macro extended functions that work on matrices will now work on the matrices stored in `r()` and `e()`, including `e(b)` and `e(V)`. These extended functions include `rownames`, `colnames`, `roweq`, `coleg`, `rowfullnames`, and `colfullnames`; see [P] [matrix define](#) and [P] [macro](#).
7. `c()` (c-class returned values) has the following new items:

item	description
<code>c(Wdays)</code>	Sun Mon ... Sat
<code>c(Weekdays)</code>	Sunday Monday Tuesday ... Saturday
<code>c(alpha)</code>	a b c d e f h j ... x y z
<code>c(ALPHA)</code>	A B C D E F H J ... X Y Z
<code>c(Mons)</code>	Jan Feb ... Dec
<code>c(Months)</code>	January February March ... December
<code>c(tracehilit)</code>	pattern to be highlighted in trace log
<code>c(maxiter)</code>	maximum iterations for maximum likelihood estimators
<code>c(varabbrev)</code>	whether variable abbreviation is on

8. A program can now be assigned properties when the program is declared, and those properties can be queried using macro extended functions. Specifically,
 - a. `program` has the new option `properties()`, which attaches properties to programs; see [P] [program](#).
 - b. A new `properties` macro extended function allows programmers to obtain the list of properties attached to a program; see [P] [macro](#).
 - c. Some of Stata's prefix commands—such as `svy` and `stepwise`—use these properties to confirm that they may be used with the command.To learn more, see [P] [program properties](#).
9. Estimation results can now be assigned properties using the new `properties()` options of `ereturn post` and `ereturn repost`. These property settings can be checked with the new function `has_eprop()`; see [P] [ereturn](#) and [D] [functions](#).
10. `m1`—Stata's maximum likelihood maximizer—has many new features, among which are
 - a. `m1` can now perform BHHH, DFP, and BFGS optimization techniques with user-defined likelihood functions. The default technique remains modified Newton–Raphson.

- b. `ml` has the new variance estimator—outer product of the gradients (OPG)—to accompany the two that were available previously —the Hessian-based estimator and the robust estimator. The new `vce()` option and the existing `robust` option specify which is used:
 - `vce(hessian)` specifies the Hessian-based variance estimator.
 - `vce(opg)` specifies the outer-product-of-the-gradients variance estimator.
 - `robust` specifies the Huber/White robust estimator.
- c. `ml` now handles “irrelevant” constraints, if specified, more elegantly. Irrelevant constraints are those that have no impact on the model. Previously, irrelevant constraints caused an error message. Now they are flagged and ignored.
 - `ml model` uses the new command `makecns` when supplied with the `constraint()` option. `makecns` generates a constraint matrix and displays a note for each constraint that causes a problem; see [P] **makecns**.
 - Prior to version 8.1, `ml` ignored the `constraint()` option if there were no predictors in the first equation. This behavior is preserved under version control. Under version 8.1, `ml` applies constraints under all circumstances.
- d. Constraints now imply a Wald test for the model chi-squared test.
- e. `mlmatbysum` is a new programmer’s command for use by method `d2` likelihood evaluators to help define the negative Hessian matrix in the case of panel-data likelihoods; see [R] **ml**.
- f. `ml` has the new subcommand `score`, which generates scores after fitting a model.
- g. `ml model` has the new option `diparm_options`, which automatically shows transformations of ancillary parameters.
- h. `ml` now saves the gradient vector in `e(gradient)`.
- i. `ml max` has the new option `shownrtolerance`, which displays the criterion value of the current iteration that is compared to `nrtolerance()`.
- j. `ml max` has the new option `search(norescale)`, which prevents any special attempts at scaling when searching for starting values.

Estimators implemented in `ml` will inherit many of these new features automatically.

For more information on these new features, see [R] **ml**. If you are programming estimators using `ml`, we highly recommend the book *Maximum Likelihood Estimation with Stata, 2nd Edition* (Gould, Pitblado, and Sribney 2003).

- 11. `ereturn post` now allows posting results without a beta vector, `e(b)`, or a covariance matrix, `e(V)`.
- 12. Stata now allows string scalars, as well as numeric scalars, so you now have two options for storing string values within programs: macros or scalars. You will want to use scalars when the string may have embedded macro-expansion characters. The maximum length of a string scalar is the same as that for a string value in the data: 80 (Intercooled Stata) or 244 (Stata/SE). See [P] **scalar**.
- 13. Significance levels can now be expressed with up to two decimal places in Stata. Before, you could not type `regress ... , level(97.5)`; now you can. `syntax` has the new option descriptor `cilevel` to make it easier to program the `level()` option. See [P] **syntax**.
- 14. `syntax` has the new `everything` option to `anything` to make the `anything` include `if`, `in`, and `using`; see [P] **syntax**.

15. `version` has the new option `born()`, which will prevent the program from running if the date of the Stata executable is earlier than the specified date. `version` also issues more descriptive error messages; see [P] **version**.
16. On Microsoft Windows and Unix platforms, the new command `window manage maintitle` allows the main title of the Stata window to be reset; see `manage maintitle` under `help window programming`.
17. `levelsof` is a new command that displays a sorted list of the distinct values of a variable. This command is especially useful for looping over the values of a variable with, say, `foreach`; see [P] **levelsof**.
18. Plugins (also known as DLLs or shared objects) written in C can now be incorporated into Stata to create new Stata commands; see [P] **plugin**.
19. The maximum number of description lines in a `stata.toc` file has been increased from 10 to 50.
20. `trace` has the new setting `set tracehilite`, which highlights a specified pattern in the trace output; see [P] **trace**.
21. The functionality of `macval()` has been extended to allow macro-dereferencing of values in a class. For example, `macval(.a.b.c)` causes the class reference `.a.b.c` to be macro-expanded only once, rather than being recursively re-expanded when the result itself contained a macro reference.
22. Variable abbreviation can now be turned on and off using the new `varabbrev` subcommand of `set` (see [R] **set**), and the setting can be made permanent so that it is remembered between sessions; see `help set varabbrev`. Note that when variable abbreviation is turned off, none of the programmer commands that process and expand varlists will recognize abbreviated variables; these commands include `unab` and `tsunab` (see [P] **unab**) and `syntax` (see [P] **syntax**).
23. The existing `confirm variable` command has the new option `exact`, which requires that the names in the specified *varlist* exactly match variable names in the dataset, disallowing the normal variable abbreviations.
24. A number of new directives and extensions to existing directives have been added to SMCL. They are summarized below within broad categories; see [P] **smcl** for complete documentation.

Jumping to marked locations in help or other files

directive	description
<code>{marker pos1}</code>	marks the current position in a file as <code>pos1</code>
<code>{help "regress##pos1"}</code>	opens the help file <code>regress.hlp</code> at the marked position <code>pos1</code>
<code>{view "my.smcl##pos1"}</code>	opens the file <code>my.smcl</code> at the marked position <code>pos1</code>

Opening help or other files in new or multiple viewers

directive	description
<code>{help "regress## mywin"}</code>	opens the help file <code>regress.hlp</code> in a new Viewer window named <code>mywin</code>
<code>{help "regress##pos1 mywin"}</code>	opens the help file <code>regress.hlp</code> at the marked position <code>pos1</code> in a new Viewer window named <code>mywin</code>
<code>{help "regress## _new"}</code>	opens the help file <code>regress.hlp</code> in a new Viewer window
<code>{view "my.smcl## mywin"}</code>	opens the file <code>my.smcl</code> in a new Viewer window named <code>mywin</code>
<code>{view "my.smcl##pos1 mywin"}</code>	opens the file <code>my.smcl</code> at the marked position <code>pos1</code> in a new Viewer window named <code>mywin</code>
<code>{view "my.smcl## _new"}</code>	opens the file <code>my.smcl</code> in a new Viewer window

Special formatting of links to help files

directive	description
<code>{helpb <i>help_topic</i>}</code>	creates link to <code>help_topic.hlp</code> , just like <code>{help <i>help_topic</i>}</code> , but displays the link in bold
<code>{helpb <i>help_topic:txt</i>}</code>	creates link to <code>help_topic.hlp</code> , just like <code>{help <i>help_topic:txt</i>}</code> , but displays <code>txt</code> in bold
<code>{manhelp <i>help_topic</i> R:txt}</code>	displays <code>[R] txt</code> and links to <code>help_topic.hlp</code>
<code>{manhelpi <i>help_topic</i> R:txt}</code>	displays <code>[R] txt</code> and links to <code>help_topic.hlp</code>

Two-column tables with indented wrapping of last column

directive	description
<code>{p2colset # # #}</code>	declares column spacing for ensuing table lines that use the <code>{p2col:...}</code> directive
<code>{p2colreset}</code>	restores default column spacing
<code>{p2col: [text]}</code>	displays <code>text</code> in column 1 and enters paragraph mode in column 2 for any text that follows, until a paragraph end is signaled; extended syntax allows columns to be specified
<code>{p2colindent: [text]}</code>	same as <code>{p2col}</code> , except <code>text</code> is output with a standard indentation for syntax-diagram-option tables
<code>{p2line}</code>	draws a line the width of the table; extended syntax allows margins around the line

New documentation conventions for syntax-diagram-option tables

directive	description
<code>{synoptset [#] [tabbed]}</code>	declares default column for syntax-diagram-option tables
<code>{synopt: [option_text]}</code>	displays <i>option_text</i> in column 1 and enters paragraph mode in column 2 for any text that follows, until a paragraph end is signaled
<code>{syntab:text}</code>	outputs text positioned as a subheading or “tab” in a syntax-diagram-option table
<code>{synopthdr [:column1_text]}</code>	displays a standard header for a syntax-diagram-option table
<code>{synoptline}</code>	draws a line extending to the boundaries of the previous <code>{synoptset}</code>

New documentation conventions for variables and varlists

directive	description
<code>{newvar}</code>	displays <i>newvar</i> while providing a link to <code>help newvar</code> ; new convention for documenting that a command accepts a new variable
<code>{varname}</code>	displays <i>varname</i> while providing a link to <code>help varname</code> ; new convention for documenting that a command accepts a variable
<code>{var}</code>	displays <i>varname</i> while providing a link to <code>help varname</code> ; abbreviated form of <code>{varname}</code>
<code>{varlist}</code>	displays <i>varlist</i> while providing a link to <code>help varlist</code> ; new convention for documenting that a command accepts a varlist
<code>{vars}</code>	displays <i>varlist</i> while providing a link to <code>help varlist</code> ; abbreviated form of <code>{varlist}</code>
<code>{depvar}</code>	displays <i>depvar</i> while providing a link to <code>help varname</code> ; new convention for documenting that a command accepts a dependent variable
<code>{depvarlist}</code>	displays <i>depvarlist</i> while providing a link to <code>help varlist</code> ; new convention for documenting that a command accepts a list of dependent variables
<code>{depvars}</code>	displays <i>depvarlist</i> while providing a link to <code>help varlist</code> ; abbreviated form of <code>{depvarlist}</code>
<code>{indepvars}</code>	displays <i>indepvars</i> while providing a link to <code>help varlist</code> ; new convention for documenting that a command accepts a list of independent variables

Note that the only change in convention is the addition of links to help files describing the syntax of variables and varlists.

Each of the above directives also accepts an optional argument that is displayed immediately after the standard display but does not otherwise change the link; for example, `{varlist:_1}` displays `varlist_1` but continues to link to `help varlist`.

Other new documentation conventions

directive	description
<code>{ifin}</code>	displays <code>[if]</code> <code>[in]</code> while providing links to <code>help if</code> and <code>help in</code> ; new convention for documenting support for <code>if</code> and <code>in</code> in a syntax diagram
<code>{weight}</code>	displays <code>[weight]</code> while providing a link to <code>help weight</code> ; new convention for documenting support for weights in a syntax diagram
<code>{dtype}</code>	displays <code>[type]</code> while providing a link to <code>help data types</code> ; new convention for documenting that a command accepts an optional data type in its syntax
<code>{dlgtab:text}</code>	displays <code>text</code> while giving it the appearance of a labeled dialog tab (extended forms support additional formatting)

Directives that simplify documenting options

directive	description
<code>{opt optname}</code>	documents options; equivalent to <code>{cmd:optname}</code>
<code>{opt opt:name}</code>	documents options that can be abbreviated; <code>{opt my:opt}</code> displays <code>myopt</code>
<code>{opt my:opt(arg)}</code>	documents options that take arguments; in this example, the option is named <code>myopt</code> and can be abbreviated <code>my—myopt(arg)</code> ; directive will correctly display arguments that are lists, such as <code>a, b, ...</code> or <code>a b c ...</code>
<code>{opth my:opt(arg)}</code>	like <code>{opt my:opt(arg)}</code> , documents options that take arguments but also provides a link to help for <code>arg</code> ; for example, <code>{opth my:opt(varlist)}</code> displays <code>myopt(varlist)</code> ; extended syntax allows the linked help to differ from the displayed argument

Directives abbreviating standard paragraph forms

directive	description
<code>{pstd}</code>	equivalent to <code>{p 4 4 2}</code>
<code>{psee}</code>	equivalent to <code>{p 4 13 2}</code>
<code>{phang}</code>	equivalent to <code>{p 4 8 2}</code>
<code>{pmore}</code>	equivalent to <code>{p 8 8 2}</code>
<code>{pin}</code>	equivalent to <code>{p 8 8 2}</code>
<code>{phang2}</code>	equivalent to <code>{p 8 12 2}</code>
<code>{pmore2}</code>	equivalent to <code>{p 12 12 2}</code>
<code>{pin2}</code>	equivalent to <code>{p 12 12 2}</code>
<code>{phang3}</code>	equivalent to <code>{p 12 16 2}</code>
<code>{pmore3}</code>	equivalent to <code>{p 16 16 2}</code>
<code>{pin3}</code>	equivalent to <code>{p 16 16 2}</code>

Other new directives and extensions to existing directives

directive	description
<code>{matacmd args [:text]}</code>	like the <code>{stata}</code> directive, but for Mata; displays <i>text</i> , and when <i>text</i> is clicked, executes the <code>mata</code> command <i>args</i>
<code>{rcenter:text}</code>	places text one space to the right when there are unequal spaces left and right
<code>{hline #}</code>	new optional #; draws a horizontal line stopping # characters from the end of the line

Help file preprocessor directive

directive	description
<code>INCLUDE help arg</code>	specifies that <code>help</code> substitute the contents of a file named <i>arg.ihelp</i> . This is useful when you need to include the same text multiple times. Note that this substitution is performed only when the file is viewed using <code>help</code> .

25. The new command `svymarkout` resets the value of a supplied 0/1 variable to 0 when any of the survey-characteristic variables set by `svyset` contain missing values; see [SVY] **svymarkout**.

26. The existing command `window manage` has the following changes and additions.

`window manage close graph [graphname | _all]`
closes the graph window named *graphname*, if it exists. If `_all` is specified, closes all graph windows.

`window manage forward graph [graphname]`
now brings the Graph window named *graphname* to the top of the other windows and otherwise works as before.

- `window manage close viewer` [*viewername* | `_all`]
closes the Viewer window named *viewername*. If `_all` is specified, closes all Viewer windows.
- `window manage forward viewer` [*viewername*]
now brings the Viewer window named *viewername* to the top of other windows and continues to work as before when no *viewername* is specified.
- `window manage minimize`
minimizes the main Stata window.
- `window manage restore`
restores the main Stata window, if it is minimized.
27. The existing command `window menu` now has the new subcommand `append_recentfiles` to add a `.dta` or `.gph` file to the **Open Recent** menu.
28. In macro expansion, double backslash (`\\`) becomes single backslash (`\`) only if the second backslash precedes macro-expansion punctuation (`'` or `$`). Previously (and still, under version control), `\\` always became `\`, so something like `\\computer\c\data` undesirably became `\computer\c\data`.

There are other new additions to Stata that will be of interest to programmers, but, because they are of interest to others, too, they are documented in [U] **1.3 What's new**.

Also See

- Complementary:** [U] **18 Programming Stata**,
[U] **1.3 What's new**,
Maximum Likelihood Estimation with Stata
- Background:** [R] **intro**