Title

stata.com

textbox_options — Options for textboxes and concept definition

Description Syntax Options Remarks and examples Also see

Description

A textbox contains one or more lines of text. The textbox options listed above specify how the text and textbox should appear.

Syntax

Textboxes contain one or more lines of text. The appearance of textboxes is controlled by the following options:

textbox_options	Description
`tstyle(`*textboxstyle*`)`	overall style
`orientation(`*orientationstyle*`)`	whether vertical or horizontal
`size(`*textsizestyle*`)`	size of text
`color(`*colorstyle*`)`	color and opacity of text
`justification(`*justificationstyle*`)`	text left, centered, right-justified
`alignment(`*alignmentstyle*`)`	text top, middle, bottom baseline
`margin(`*marginstyle*`)`	margin from text to border
`linegap(`*relativesize*`)`	space between lines
`width(`*relativesize*`)`	width of textbox override
`height(`*relativesize*`)`	height of textbox override
`box` or `nobox`	whether border is drawn around box
`bcolor(`*colorstyle*`)`	color and opacity of background and border
`fcolor(`*colorstyle*`)`	color and opacity of background
`lstyle(`*linestyle*`)`	overall style of border
`lpattern(`*linepatternstyle*`)`	line pattern of border
`lwidth(`*linewidthstyle*`)`	thickness of border
`lcolor(`*colorstyle*`)`	color and opacity of border
`bmargin(`*marginstyle*`)`	margin from border outwards
`bexpand`	expand box in direction of text
`placement(`*compassdirstyle*`)`	location of textbox override

The above options invariably occur inside other options. For instance, the syntax of `title()` (see [G-3] ***title_options***) is

`title("`*string*`" [ "`*string*`" [ ... ] ] [ , `*title_options textbox_options*` ])`

so any of the options above can appear inside the `title()` option:

```
. graph ..., ... title("My title", color(green) box) ...
```

Options

`tstyle(`*textboxstyle*`)` specifies the overall style of the textbox. Think of a textbox as a set of characteristics that include, in addition to the text, the size of font, the color, whether lines are drawn around the box, etc. The *textboxstyle* you choose specifies all of those things, and it is from there that the changes you make by specifying the other operations take effect.

The default is determined by the overall context of the text (such as whether it is due to `title()`, `subtitle()`, etc.), and that in turn is specified by the scheme (see [G-4] **schemes intro**). That is, identifying the name of the default style in a context is virtually impossible.

Option `tstyle()` is rarely specified. Usually, you simply let the overall style be whatever it is and specify the other textbox options to modify it. Do not, however, dismiss the idea of looking for a better overall style that more closely matches your desires.

See [G-4] ***textboxstyle***.

`orientation(`*orientationstyle*`)` specifies whether the text and box are to be oriented horizontally or vertically (text reading from bottom to top or text reading from top to bottom). See [G-4] ***orientationstyle***.

`size(`*textsizestyle*`)` specifies the size of the text that appears inside the textbox. See [G-4] ***textsizestyle***.

`color(`*colorstyle*`)` specifies the color and opacity of the text that appears inside the textbox. See [G-4] ***colorstyle***.

`justification(`*justificationstyle*`)` specifies how the text is to be "horizontally" aligned in the box. Choices include `left`, `right`, and `center`. Think of the textbox as being horizontal, even if it is vertical when specifying this option. See [G-4] ***justificationstyle***.

`alignment(`*alignmentstyle*`)` specifies how the text is to be "vertically" aligned in the box. Choices include `baseline`, `middle`, and `top`. Think of the textbox as being horizontal, even if it is vertical when specifying this option. See [G-4] ***alignmentstyle***.

`margin(`*marginstyle*`)` specifies the margin around the text (the distance from the text to the borders of the box). The text that appears in a box, plus `margin()`, determine the overall size of the box. See [G-4] ***marginstyle***.

When dealing with rotated textboxes—textboxes for which `orientation(vertical)` or `orientation(rvertical)` has been specified—the margins for the left, right, bottom, and top refer to the margins before rotation.

`linegap(`*relativesize*`)` specifies the distance between lines. See [G-4] ***relativesize*** for argument choices.

`width(`*relativesize*`)` and `height(`*relativesize*`)` override Stata's usual determination of the width and height of the textbox on the basis of its contents. See *Width and height* under *Remarks and examples* below. See [G-4] ***relativesize*** for argument choices.

`box` and `nobox` specify whether a box is to be drawn outlining the border of the textbox. The default is determined by the `tstyle()`, which in turn is determined by context, etc. In general, the default is not to outline boxes, so the option to outline boxes is `box`. If an outline appears by default, `nobox` is the option to suppress the outlining of the border. No matter what the default, you can specify `box` or `nobox`.

`bcolor(`*colorstyle*`)` specifies the color and opacity of both the background of the box and the outlined border. This option is typically not specified because it results in the border disappearing into the background of the textbox; see options `fcolor()` and `lcolor()` below for alternatives. The color matters only if `box` is also specified; otherwise, `bcolor()` is ignored. See [G-4] ***colorstyle*** for a list of color choices.

`fcolor(`*colorstyle*`)` specifies the color and opacity of the background of the box. The background of the box is filled with the `fcolor()` only if `box` is also specified; otherwise, `fcolor()` is ignored. See [G-4] ***colorstyle*** for a list of color choices.

`lstyle(`*linestyle*`)` specifies the overall style of the line used to outline the border. The style includes the line's pattern (solid, dashed, etc.), thickness, and color.

You need not specify `lstyle()` just because there is something you want to change about the look of the line. Options `lpattern`, `lwidth()`, and `lcolor()` will allow you to change the attributes individually. You specify `lstyle()` when there is a style that is exactly what you desire or when another style would allow you to specify fewer changes.

See [G-4] ***linestyle*** for a list of style choices and see [G-4] **concept: lines** for a discussion of lines in general.

`lpattern(`*linepatternstyle*`)` specifies the pattern of the line outlining the border. See [G-4] ***linepatternstyle***. Also see [G-4] **concept: lines** for a discussion of lines in general.

`lwidth(`*linewidthstyle*`)` specifies the thickness of the line outlining the border. See [G-4] ***linewidthstyle***. Also see [G-4] **concept: lines** for a discussion of lines in general.

`lcolor(`*colorstyle*`)` specifies the color and opacity of the border of the box. The border color matters only if `box` is also specified; otherwise, the `lcolor()` is ignored. See [G-4] ***colorstyle*** for a list of color choices.

`bmargin(`*marginstyle*`)` specifies the margin between the border and the containing box. See [G-4] ***marginstyle***.

`bexpand` specifies that the textbox be expanded in the direction of the text, made wider if the text is horizontal, and made longer if the text is vertical. It is expanded to the borders of its containing box. See [G-3] ***title_options*** for a demonstration of this option.

`placement(`*compassdirstyle*`)` overrides default placement; see *Appendix: Overriding default or context-specified positioning* below. See [G-4] ***compassdirstyle*** for argument choices.

Remarks and examples

stata.com

Remarks are presented under the following headings:

Definition of a textbox

A textbox is one or more lines of text

single-line textbox

1st line of multiple-line textbox
2nd line of multiple-line textbox

for which the borders may or may not be visible (controlled by the `box`/`nobox` option). Textboxes can be horizontal or vertical

horizontal

vertical

rvertical

in an `orientation(vertical)`	*in an* `orientation(rvertical)`
textbox, letters are rotated	*textbox, letters are rotated*
90 degrees counterclockwise;	*90 degrees clockwise;*
`orientation(vertical)` *reads*	`orientation(rvertical)` *reads*
bottom to top	*top to bottom*

Even in vertical textboxes, options are stated in horizontal terms of left and right. Think horizontally, and imagine the rotation as being performed at the end.

Position

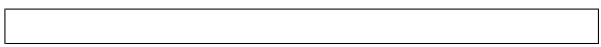
Textboxes are first formed and second positioned on the graph. The *textbox_options* affect the construction of the textbox, not its positioning. The options that control its positioning are provided by the context in which the textbox is used. For instance, the syntax of the `title()` option—see [G-3] ***title_options***—is

`title("`*string*`" ... [ , position(...) ring(...) span(...) ...`*textbox_options*` ])`

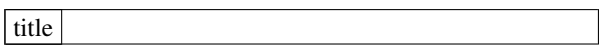
`title()`'s `position()`, `ring()`, and `span()` options determine where the title (that is, textbox) is positioned. Once the textbox is formed, its contents no longer matter; it is just a box to be positioned on the graph.

Textboxes are positioned inside other boxes. For instance, the textbox might be

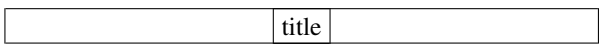
and, because of the `position()`, `ring()`, and `span()` options specified, `title()` might position that box somewhere on the top "line":



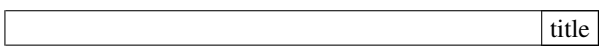
There are many ways the smaller box could be fit into the larger box, which is the usual case, and forgive us for combining two discussions: how boxes fit into each other and the controlling of placement. If you specified `title()`'s `position(11)` option, the result would be



If you specified `title()`'s `position(12)` option, the result would be



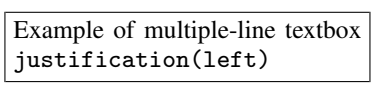
If you specified `title()`'s `position(1)` option, the result would be

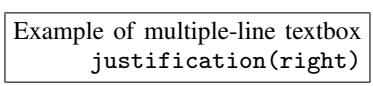


Justification

An implication of the above is that it is not the textbox option `justification()` that determines whether the title is centered; it is `title()`'s `position()` option.

Remember, textbox options describe the construction of textboxes, not their use. `justification(left | right | center)` determines how text is placed in multiple-line textboxes:





Example of multiple-line textbox `justification(center)`

Textboxes are no wider than the text of their longest line. `justification()` determines how lines shorter than the longest are placed inside the box. In a one-line textbox,

single-line textbox

it does not matter how the text is justified.

Position and justification combined

With positioning options provided by the context in which the textbox is being used, and the `justification()` option, you can create many different effects in the presentation of multiple-line textboxes. For instance, considering `title()`, you could produce

	First line of title Second line	

(1)

or

	First line of title Second line	

(2)

or

	First line of title Second line	

(3)

or

	First line of title Second line

(4)

or

	First line of title Second line

(5)

or

	First line of title Second line

(6)

or many others. The corresponding commands would be

```
. graph ..., title("First line of title" "Second line",       (1)
                              position(12) justification(left))
. graph ..., title("First line of title" "Second line",       (2)
                              position(12) justification(center))
. graph ..., title("First line of title" "Second line",       (3)
                              position(12) justification(right))
. graph ..., title("First line of title" "Second line",       (4)
                              position(1) justification(left))
. graph ..., title("First line of title" "Second line",       (5)
                              position(1) justification(center))
. graph ..., title("First line of title" "Second line",       (6)
                              position(1) justification(right))
```

Margins

There are two margins: `margin()` and `bmargin()`. `margin()` specifies the margin between the text and the border. `bmargin()` specifies the margin between the border and the containing box.

By default, textboxes are the smallest rectangle that will just contain the text. If you specify `margin()`, you add space between the text and the borders of the bounding rectangle:

`margin(zero)` textbox

textbox with ample margin on all four sides

`margin(`*marginstyle*`)` allows different amounts of padding to be specified above, below, left, and right of the text; see [G-4] ***marginstyle***. `margin()` margins make the textbox look better when the border is outlined via the `box` option and/or the box is shaded via the `bcolor()` or `fcolor()` option.

`bmargin()` margins are used to move the textbox a little or a lot when the available positioning options are inadequate. Consider specifying the `caption()` option (see [G-3] ***title_options***) so that it is inside the plot region:

```
. graph ..., caption("My caption", ring(0) position(7))
```

Seeing the result, you decide that you want to shift the box up and to the right a bit:

```
. graph ..., caption("My caption", ring(0) position(7)
             bmargin("2 0 2 0"))
```

The `bmargin()` numbers (and `margin()` numbers) are the top, bottom, left, and right amounts, and the amounts are specified as relative sizes (see [G-4] ***relativesize***). We specified a 2% bottom margin and a 2% left margin, thus pushing the caption box up and to the right.

Width and height

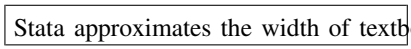
The width and the height of a textbox are determined by its contents (the text width and number of lines) plus the margins just discussed. The width calculation, however, is based on an approximation, with the result that the textbox that should look like this

Stata approximates the width of textboxes

can end up looking like this

Stata approximates the width of textboxes

or like this



You will not notice this problem unless the borders are being drawn (option `box`) because, without borders, in all three cases you would see

Stata approximates the width of textboxes

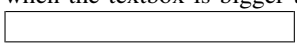
For an example of this problem and the solution, see *Use of the textbox option width()* in [G-3] ***added_text_options***. If the problem arises, use `width(`*relativesize*`)` to work around it. Getting the `width()` right is a matter of trial and error. The correct width will nearly always be between 0 and 100.

Corresponding to `width(`*relativesize*`)`, there is `height(`*relativesize*`)`. This option is less useful because Stata never gets the height incorrect.

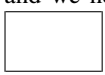
Appendix: Overriding default or context-specified positioning

What follows is really a footnote. We said previously that where a textbox is located is determined by the context in which it is used and by the positioning options provided by that context. Sometimes you wish to override that default, or the context may not provide such control. In such cases, the option `placement()` allows you to take control.

Let us begin by correcting a misconception we introduced. We previously said that textboxes are fit inside other boxes when they are positioned. That is not exactly true. For instance, what happens when the textbox is bigger than the box into which it is being placed? Say that we have the textbox

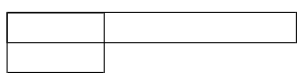


and we need to put it "in" the box



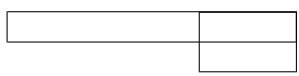
The way things work, textboxes are not put inside other boxes; they are merely positioned so that they align a certain way with the preexisting box. Those alignment rules are such that, if the preexisting box is larger than the textbox, the result will be what is commonly meant by "inside". The alignment rules are either to align one of the four corners or to align and center on one of the four edges.

In the example just given, the textbox could be positioned so that its northwest corner is coincident with the northwest corner of the preexisting box,



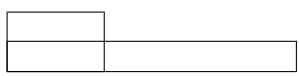
`placement(nw)`

or so that their northeast corners are coincident,



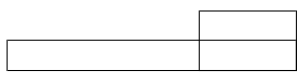
`placement(ne)`

or so that their southwest corners are coincident,



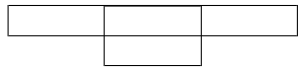
`placement(sw)`

or so that their southeast corners are coincident,



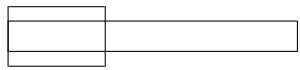
`placement(se)`

or so that the midpoint of the top edges are the same,



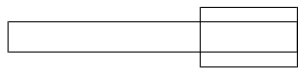
`placement(n)`

or so that the midpoint of the left edges are the same,



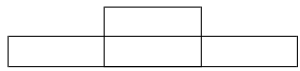
`placement(w)`

or so that the midpoint of the right edges are the same,



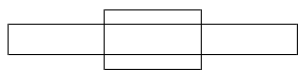
`placement(e)`

or so that the midpoint of the bottom edges are the same,



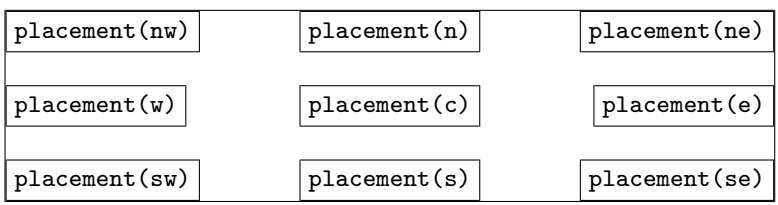
`placement(s)`

or so that the center point of the boxes are the same:



`placement(c)`

If you have trouble seeing any of the above, consider what you would obtain if the preexisting box were larger than the textbox. Below we show the preexisting box with eight different textboxes:



Also see

[G-4] ***alignmentstyle*** — Choices for vertical alignment of text

[G-4] ***colorstyle*** — Choices for color

[G-4] ***compassdirstyle*** — Choices for location

[G-4] ***justificationstyle*** — Choices for how text is justified

[G-4] ***linepatternstyle*** — Choices for whether lines are solid, dashed, etc.

[G-4] ***linewidthstyle*** — Choices for thickness of lines

[G-4] ***marginstyle*** — Choices for size of margins

[G-4] ***orientationstyle*** — Choices for orientation of textboxes

[G-4] ***relativesize*** — Choices for sizes of objects

[G-4] ***text*** — Text in graphs

[G-4] ***textboxstyle*** — Choices for the overall look of text including border

[G-4] ***textsizestyle*** — Choices for the size of text

[G-3] ***title_options*** — Options for specifying titles