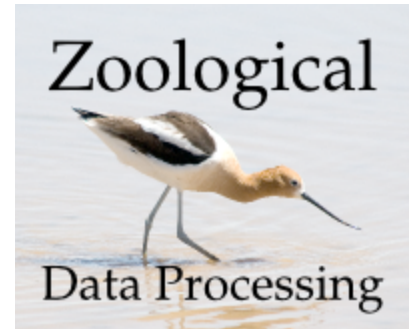


[Next](#) / [Previous](#) / [Contents](#) / [Shipman's homepage](#)

***Tkinter* 8.5 reference: a GUI for Python**



8. The Canvas widget

A canvas is a rectangular area intended for drawing pictures or other complex layouts. On it you can place graphics, text, widgets, or frames. See the following sections for methods that create objects on canvases:

- `.create_arc()`: A slice out of an ellipse. See [Section 8.7, “Canvas arc objects”](#).
- `.create_bitmap()`: An image as a bitmap. See [Section 8.8, “Canvas bitmap objects”](#).
- `.create_image()`: A graphic image. See [Section 8.9, “Canvas image objects”](#).
- `.create_line()`: One or more line segments. See [Section 8.10, “Canvas line objects”](#).
- `.create_oval()`: An ellipse; use this also for drawing circles, which are a special case of an ellipse. See [Section 8.11, “Canvas oval objects”](#).

- `.create_polygon()`: A polygon. See [Section 8.12, “Canvas polygon objects”](#).
- `.create_rectangle()`: A rectangle. See [Section 8.13, “Canvas rectangle objects”](#).
- `.create_text()`: Text annotation. See [Section 8.14, “Canvas text objects”](#).
- `.create_window()`: A rectangular window. See [Section 8.15, “Canvas window objects”](#).

To create a Canvas object:

```
w = tk.Canvas(parent, option=value, ...)
```

The constructor returns the new Canvas widget. Supported options include:

Table 6. Canvas widget options

bd or borderwidth	Width of the border around the outside of the canvas; see Section 5.1, “Dimensions” . The default is two pixels.
bg or background	Background color of the canvas. Default is a light gray, about '#E4E4E4'.
closeenough	A float that specifies how close the mouse must be to an item to be considered inside it. Default is 1.0.
confine	If true (the default), the canvas cannot be scrolled outside of the scrollregion (see below).

<code>cursor</code>	Cursor used in the canvas. See Section 5.8, “Cursors” .
<code>height</code>	Size of the canvas in the Y dimension. See Section 5.1, “Dimensions” .
<code>highlightbackground</code>	Color of the focus highlight when the widget does not have focus. See Section 53, “Focus: routing keyboard input” .
<code>highlightcolor</code>	Color shown in the focus highlight.
<code>highlightthickness</code>	Thickness of the focus highlight. The default value is 1.
<code>relief</code>	The relief style of the canvas. Default is tk.FLAT. See Section 5.6, “Relief styles” .
<code>scrollregion</code>	A tuple (<i>w</i> , <i>n</i> , <i>e</i> , <i>s</i>) that defines over how large an area the canvas can be scrolled, where <i>w</i> is the left side, <i>n</i> the top, <i>e</i> the right side, and <i>s</i> the bottom.
<code>selectbackground</code>	The background color to use displaying selected items.
<code>selectborderwidth</code>	The width of the border to use around selected items.
<code>selectforeground</code>	The foreground color to use displaying selected items.
<code>takefocus</code>	Normally, focus (see Section 53, “Focus: routing keyboard input”) will cycle through this widget with the tab key only if there are keyboard bindings set for it (see Section 54, “Events” for

	an overview of keyboard bindings). If you set this option to 1, focus will always visit this widget. Set it to ' ' to get the default behavior.
width	Size of the canvas in the X dimension. See Section 5.1, “Dimensions” .
xscrollincrement	Normally, canvases can be scrolled horizontally to any position. You can get this behavior by setting xscrollincrement to zero. If you set this option to some positive dimension , the canvas can be positioned only on multiples of that distance, and the value will be used for scrolling by <i>scrolling units</i> , such as when the user clicks on the arrows at the ends of a scrollbar. For more information on scrolling units, see Section 22, “The Scrollbar widget” .
xscrollcommand	If the canvas is scrollable, set this option to the .set() method of the horizontal scrollbar.
yscrollincrement	Works like xscrollincrement, but governs vertical movement.
yscrollcommand	If the canvas is scrollable, this option should be the .set() method of the vertical scrollbar.

8.1. Canvas coordinates

Because the canvas may be larger than the window, and

equipped with scrollbars to move the overall canvas around in the window, there are two coordinate systems for each canvas:

- The *window coordinates* of a point are relative to the top left corner of the area on the display where the canvas appears.
- The *canvas coordinates* of a point are relative to the top left corner of the total canvas.

Next: [8.2. The Canvas display list](#)

Contents: [Tkinter 8.5 reference: a GUI for Python](#)

Previous: [7. The Button widget](#)

Home: [John Shipman's Home Sweet Homepage](#)

John W. Shipman

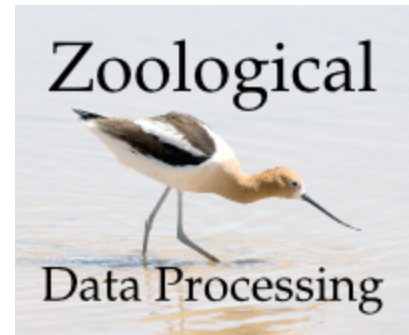
Comments welcome: john@nmt.edu

Last updated: 2013-09-09 22:58

URL: <http://www.nmt.edu/~shipman/soft/tkinter/web/canvas.html>

[Next](#) / [Previous](#) / [Contents](#) / [Shipman's homepage](#)

***Tkinter* 8.5 reference: a GUI for Python**



8.2. The Canvas display list

The *display list* refers to the sequence of all the objects on the canvas, from background (the “bottom” of the display list) to foreground (the “top”).

If two objects overlap, the one *above* the other in the display list means the one closer to the foreground, which will appear in the area of overlap and obscure the one *below*. By default, new objects are always created at the top of the display list (and hence in front of all other objects), but you can re-order the display list.

Next: [8.3. Canvas object IDs](#)

Contents: [Tkinter 8.5 reference: a GUI for Python](#)

Previous: [8. The Canvas widget](#)

Home: [John Shipman's Home Sweet Homepage](#)

John W. Shipman

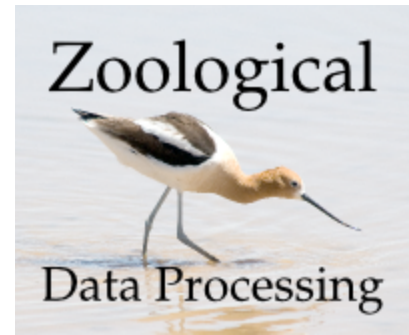
Comments welcome: john@nmt.edu

Last updated: 2013-09-09 22:58

URL: <http://www.nmt.edu/~shipman/soft/tkinter/web/display-list.html>

[Next](#) / [Previous](#) / [Contents](#) / [Shipman's homepage](#)

***Tkinter* 8.5 reference: a GUI for Python**



8.3. Canvas object IDs

The *object ID* of an object on the canvas is the value returned by the constructor for that object. All object ID values are simple integers, and the object ID of an object is unique within that canvas.

Next: [8.4. Canvas tags](#)

Contents: [Tkinter 8.5 reference: a GUI for Python](#)

Previous: [8.2. The Canvas display list](#)

Home: [John Shipman's Home Sweet Homepage](#)

John W. Shipman

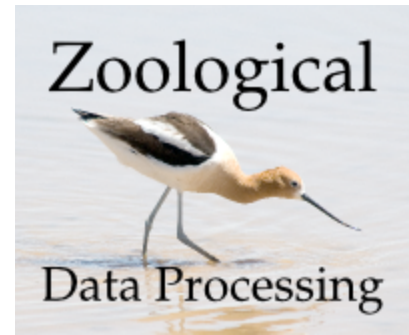
Comments welcome: john@nmt.edu

Last updated: 2013-09-09 22:58

URL: <http://www.nmt.edu/~shipman/soft/tkinter/web/canvas-oid.html>

[Next](#) / [Previous](#) / [Contents](#) / [Shipman's homepage](#)

***Tkinter* 8.5 reference: a GUI for Python**



8.4. Canvas tags

A *tag* is a string that you can associate with objects on the canvas.

- A tag can be associated with any number of objects on the canvas, including zero.
- An object can have any number of tags associated with it, including zero.

Tags have many uses. For example, if you are drawing a map on a canvas, and there are text objects for the labels on rivers, you could attach the tag 'riverLabel' to all those text objects. This would allow you to perform operations on all the objects with that tag, such as changing their color or deleting them.

Next: [8.5. Canvas *tagOrId* arguments](#)

Contents: [Tkinter 8.5 reference: a GUI for Python](#)

Previous: [8.3. Canvas object IDs](#)

Home: [John Shipman's Home Sweet Homepage](#)

John W. Shipman

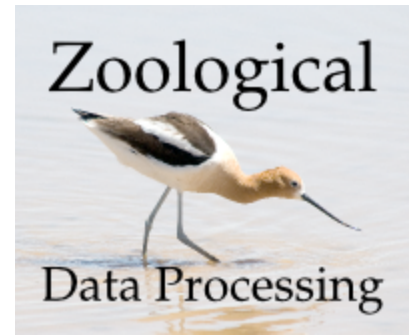
Comments welcome: john@nmt.edu

Last updated: 2013-09-09 22:58

URL: <http://www.nmt.edu/~shipman/soft/tkinter/web/canvas-tags.html>

[Next](#) / [Previous](#) / [Contents](#) / [Shipman's homepage](#)

***Tkinter* 8.5 reference: a GUI for Python**



8.5. Canvas *tagOrId* arguments

A `tagOrId` argument specifies one or more objects on the canvas.

- If a `tagOrId` argument is an integer, it is treated as an object ID, and it applies only to the unique object with that ID. See [Section 8.3, “Canvas object IDs”](#).
- If such an argument is a string, it is interpreted as a tag, and selects all the objects that have that tag (if there are any). See [Section 8.4, “Canvas tags”](#).

Next: [8.6. Methods on Canvas widgets](#)

Contents: [Tkinter 8.5 reference: a GUI for Python](#)

Previous: [8.4. Canvas tags](#)

Home: [John Shipman's Home Sweet Homepage](#)

John W. Shipman

Comments welcome: john@nmt.edu

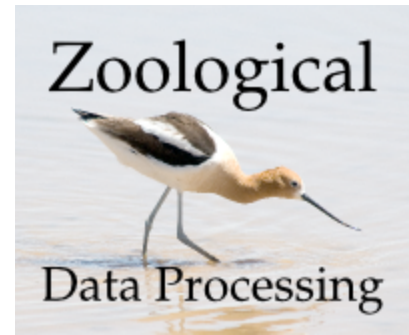
Last updated: 2013-09-09 22:58

URL: <http://www.nmt.edu/~shipman/soft/tkinter/web/tag-or-id.html>

[Next](#) / [Previous](#) / [Contents](#) / [Shipman's homepage](#)

***Tkinter* 8.5**

reference: a GUI for Python



8.6. Methods on Canvas widgets

All Canvas objects support these methods:

`.addtag_above(newTag, tagOrId)`

Attaches a new [tag](#) to the object just above the one specified by [tagOrId](#) in the display list. The *newTag* argument is the tag you want to attach, as a string.

`.addtag_all(newTag)`

Attaches the given [tag](#) *newTag* to all the objects on the canvas.

`.addtag_below(newTag, tagOrID)`

Attaches a new [tag](#) to the object just below the one specified by [tagOrId](#) in the display list. The *newTag* argument is a tag string.

`.addtag_closest(newTag, x, y, halo=None, start=None)`

Adds a [tag](#) to the object closest to screen coordinate (x,y). If there are two or more objects at the same distance, the one higher in the [display list](#) is selected.

Use the *halo* argument to increase the effective size of the point. For example, a value of 5 would treat any object within 5 pixels of (x,y) as overlapping.

If an [object ID](#) is passed in the *start* argument, this method tags the highest qualifying object that is below *start* in the display list.

.addtag_enclosed(*newTag*, *x1*, *y1*, *x2*, *y2*)

Add [tag](#) *newTag* to all objects that occur completely within the rectangle whose top left corner is (*x1*, *y1*) and whose bottom right corner is (*x2*, *y2*).

.addtag_overlapping(*newTag*, *x1*, *y1*, *x2*, *y2*)

Like the previous method, but affects all objects that share at least one point with the given rectangle.

.addtag_withtag(*newTag*, *tag0rId*)

Adds [tag](#) *newTag* to the object or objects specified by [tag0rId](#).

.bbox(*tag0rId*=None)

Returns a tuple (*x1*, *y1*, *x2*, *y2*) describing a rectangle that encloses all the objects specified by [tag0rId](#). If the argument is omitted, returns a rectangle enclosing all objects on the canvas. The top left corner of the rectangle is (*x1*, *y1*) and the bottom right corner is (*x2*, *y2*).

.canvasx(*screenx*, *gridspacing*=None)

Translates a window x coordinate *screenx* to a canvas coordinate. If *gridspacing* is supplied, the canvas coordinate is rounded to the nearest multiple of that value.

.canvasy(*screeny*, gridspacing=None)

Translates a window y coordinate *screeny* to a canvas coordinate. If *gridspacing* is supplied, the canvas coordinate is rounded to the nearest multiple of that value.

.coords(*tagOrId*, *x0*, *y0*, *x1*, *y1*, ..., *xn*, *yn*)

If you pass only the [tagOrId](#) argument, returns a tuple of the coordinates of the lowest or only object specified by that argument. The number of coordinates depends on the type of object. In most cases it will be a 4-tuple (*x1*, *y1*, *x2*, *y2*) describing the bounding box of the object.

You can move an object by passing in new coordinates.

.dchars(*tagOrId*, first=0, last=first)

Deletes characters from a text item or items. Characters between *first* and *last* *inclusive* are deleted, where those values can be integer indices or the string 'end' to mean the end of the text. For example, for a canvas *C* and an item *I*, *C.dchars(I, 1, 1)* will remove the second character.

.delete(*tagOrId*)

Deletes the object or objects selected by [tagOrId](#). It is not considered an error if no items match *tagOrId*.

.dtag(*tagOrId*, tagToDelete)

Removes the [tag](#) specified by *tagToDelete* from the object or objects specified by [tagOrId](#).

.find_above(*tagOrId*)

Returns the ID number of the object just above the object

specified by [tagOrId](#). If multiple objects match, you get the highest one. Returns an empty tuple if you pass it the object ID of the highest object.

.find_all()

Returns a list of the [object ID numbers](#) for all objects on the canvas, from lowest to highest.

.find_below(tagOrId)

Returns the [object ID](#) of the object just below the one specified by [tagOrId](#). If multiple objects match, you get the lowest one. Returns an empty tuple if you pass it the object ID of the lowest object.

.find_closest(x, y, halo=None, start=None)

Returns a singleton tuple containing the [object ID](#) of the object closest to point (x, y). If there are no qualifying objects, returns an empty tuple.

Use the *halo* argument to increase the effective size of the point. For example, *halo*=5 would treat any object within 5 pixels of (x, y) as overlapping.

If an object ID is passed as the *start* argument, this method returns the highest qualifying object that is below *start* in the [display list](#).

.find_enclosed(x1, y1, x2, y2)

Returns a list of the [object IDs](#) of all objects that occur completely within the rectangle whose top left corner is (x1, y1) and bottom right corner is (x2, y2).

.find_overlapping(x1, y1, x2, y2)

Like the previous method, but returns a list of the object IDs of all the objects that share at least one point with the given rectangle.

.find_withtag(*tagOrId*)

Returns a list of the [object IDs](#) of the object or objects specified by [tagOrId](#).

.focus(*tagOrId=None*)

Moves the focus to the object specified by [tagOrId](#). If there are multiple such objects, moves the focus to the first one in the [display list](#) that allows an insertion cursor. If there are no qualifying items, or the canvas does not have focus, focus does not move.

If the argument is omitted, returns the ID of the object that has focus, or ' ' if none of them do.

.gettags(*tagOrId*)

If [tagOrId](#) is an object ID, returns a list of all the tags associated with that object. If the argument is a [tag](#), returns all the tags for the lowest object that has that tag.

.icursor(*tagOrId*, *index*)

Assuming that the selected item allows text insertion and has the focus, sets the insertion cursor to *index*, which may be either an integer index or the string 'end'. Has no effect otherwise.

.index(*tagOrId*, *specifier*)

Returns the integer index of the given *specifier* in the text item specified by [tagOrId](#) (the lowest one that, if *tagOrId* specifies multiple objects). The return value is the

corresponding position as an integer, with the usual Python convention, where 0 is the position before the first character.

The *specifier* argument may be any of:

- `tk.INSERT`, to return the current position of the insertion cursor.
- `tk.END`, to return the position after the last character of the item.
- `tk.SEL_FIRST`, to return the position of the start of the current text selection. *Tkinter* will raise a `tk.TclError` exception if the text item does not currently contain the text selection.
- `tk.SEL_LAST`, to return the position after the end of the current text selection, or raise `tk.TclError` if the item does not currently contain the selection.
- A string of the form “@*x,y*”, to return the character of the character containing canvas coordinates (*x*, *y*). If those coordinates are above or to the left of the text item, the method returns 0; if the coordinates are to the right of or below the item, the method returns the index of the end of the item.

`.insert(tagOrId, specifier, text)`

Inserts the given *string* into the object or objects specified by [*tagOrId*](#), at the position given by the *specifier* argument.

The *specifier* values may be:

- Any of the keywords `tk.INSERT`, `tk.END`, `tk.SEL_FIRST`, or `tk.SEL_LAST`. Refer to the description of the index

method above for the interpretation of these codes.

- The position of the desired insertion, using the normal Python convention for positions in strings.

.itemcget(*tagOrId*, *option*)

Returns the value of the given configuration *option* in the selected object (or the lowest object if [*tagOrId*](#) specifies more than one). This is similar to the `.cget()` method for *Tkinter* objects.

.itemconfigure(*tagOrId*, *option*, ...)

If no *option* arguments are supplied, returns a dictionary whose keys are the options of the object specified by [*tagOrId*](#) (the lowest one, if *tagOrId* specifies multiple objects).

To change the configuration option of the specified item, supply one or more keyword arguments of the form *option=value*.

.move(*tagOrId*, *xAmount*, *yAmount*)

Moves the items specified by [*tagOrId*](#) by adding *xAmount* to their x coordinates and *yAmount* to their y coordinates.

.postscript(*option*, ...)

Generates an Encapsulated PostScript representation of the canvas's current contents. The options include:

colormode	Use 'color' for color output, 'gray' for grayscale, or 'mono' for black and white.
file	If supplied, names a file where the PostScript

	will be written. If this option is not given, the PostScript is returned as a string.
height	How much of the Y size of the canvas to print. Default is the entire visible height of the canvas.
rotate	If false, the page will be rendered in portrait orientation; if true, in landscape.
x	Leftmost canvas coordinate of the area to print.
y	Topmost canvas coordinate of the area to print.
width	How much of the X size of the canvas to print. Default is the visible width of the canvas.

`.scale(tagOrId, xOffset, yOffset, xScale, yScale)`

Scale all objects according to their distance from a point $P=(xOffset, yOffset)$. The scale factors *xScale* and *yScale* are based on a value of 1.0, which means no scaling. Every point in the objects selected by [tagOrId](#) is moved so that its x distance from P is multiplied by *xScale* and its y distance is multiplied by *yScale*.

This method will not change the size of a text item, but may move it.

`.scan_dragto(x, y, gain=10.0)`

See the `.scan_mark()` method below.

`.scan_mark(x, y)`

This method is used to implement fast scrolling of a canvas. The intent is that the user will press and hold a

mouse button, then move the mouse up to scan (scroll) the canvas horizontally and vertically in that direction at a rate that depends on how far the mouse has moved since the mouse button was depressed.

To implement this feature, bind the mouse's button-down event to a handler that calls `scan_mark(x, y)` where `x` and `y` are the current mouse coordinates. Bind the `<Motion>` event to a handler that, assuming the mouse button is still down, calls `scan_dragto(x, y, gain)` where `x` and `y` are the current mouse coordinates.

The *gain* argument controls the rate of scanning. This argument has a default value of 10.0. Use larger numbers for faster scanning.

`.select_adjust(oid, specifier)`

Adjusts the boundaries of the current text selection to include the position given by the *specifier* argument, in the text item with the [object ID](#) *oid*.

The current selection anchor is also set to the specified position. For a discussion of the selection anchor, see the canvas `select_from` method below.

For the values of *specifier*, see the canvas `insert` method above.

`.select_clear()`

Removes the current text selection, if it is set. If there is no current selection, does nothing.

`.select_from(oid, specifier)`

This method sets the *selection anchor* to the position given by the *specifier* argument, within the text item whose

[object ID](#) is given by *oid*.

The currently selected text on a given canvas is specified by three positions: the start position, the end position, and the selection anchor, which may be anywhere within those two positions.

To change the position of the currently selected text, use this method in combination with the `select_adjust`, `select_from`, and `select_to` canvas methods (q.v.).

`.select_item()`

If there is a current text selection on this canvas, return the [object ID](#) of the text item containing the selection. If there is no current selection, this method returns `None`.

`.select_to(oid, specifier`

This method changes the current text selection so that it includes the select anchor and the position given by *specifier* within the text item whose [object ID](#) is given by *oid*. For the values of *specifier*, see the canvas `insert` method above.

`.tag_bind(tagOrId, sequence=None, function=None, add=None)`

Binds events to objects on the canvas. For the object or objects selected by *tagOrId*, associates the handler *function* with the event *sequence*. If the *add* argument is a string starting with '+', the new binding is added to existing bindings for the given *sequence*, otherwise the new binding replaces that for the given *sequence*.

For general information on event bindings, see [Section 54, “Events”](#).

Note that the bindings are applied to items that have this

tag at the time of the `tag_bind` method call. If tags are later removed from those items, the bindings will persist on those items. If the tag you specify is later applied to items that did not have that tag when you called `tag_bind`, that binding will *not* be applied to the newly tagged items.

`.tag_lower(tagOrId, belowThis)`

Moves the object or objects selected by [*tagOrId*](#) within the [display list](#) to a position just below the first or only object specied by the tag or ID *belowThis*.

If there are multiple items with tag *tagOrId*, their relative stacking order is preserved.

This method does not affect canvas window items. To change a window item's stacking order, use a `lower` or `lift` method on the window.

`.tag_raise(tagOrId, aboveThis)`

Moves the object or objects selected by [*tagOrId*](#) within the [display list](#) to a position just above the first or only object specied by the tag or ID *aboveThis*.

If there are multiple items with tag *tagOrId*, their relative stacking order is preserved.

This method does not affect canvas window items. To change a window item's stacking order, use a `lower` or `lift` method on the window.

`.tag_unbind(tagOrId, sequence, funcId=None)`

Removes bindings for handler *funcId* and event *sequence* from the canvas object or objects specified by [*tagOrId*](#). See [Section 54, “Events”](#).

.type(*tagOrId*)

Returns the type of the first or only object specified by [*tagOrId*](#). The return value will be one of the strings 'arc', 'bitmap', 'image', 'line', 'oval', 'polygon', 'rectangle', 'text', or 'window'.

.xview(tk.MOVETO, *fraction*)

This method scrolls the canvas relative to its image, and is intended for binding to the command option of a related scrollbar. The canvas is scrolled horizontally to a position given by *offset*, where 0.0 moves the canvas to its leftmost position and 1.0 to its rightmost position.

.xview(tk.SCROLL, *n*, *what*)

This method moves the canvas left or right: the *what* argument specifies how much to move and can be either tk.UNITS or tk.PAGES, and *n* tells how many units to move the canvas to the right relative to its image (or left, if negative).

The size of the move for tk.UNITS is given by the value of the canvas's xscrollincrement option; see [Section 22, “The Scrollbar widget”](#).

For movements by tk.PAGES, *n* is multiplied by nine-tenths of the width of the canvas.

.xview_moveto(*fraction*)

This method scrolls the canvas in the same way as .xview(tk.MOVETO, *fraction*).

.xview_scroll(*n*, *what*)

Same as .xview(tk.SCROLL, *n*, *what*).

`.yview(tk.MOVETO, fraction)`

The vertical scrolling equivalent of `.xview(tk.MOVETO,...)`.

`.yview(tk.SCROLL, n, what)`

The vertical scrolling equivalent of `.xview(tk.SCROLL,...)`.

`.yview_moveto(fraction)`

The vertical scrolling equivalent of `.xview()`.

`.yview_scroll(n, what)`

The vertical scrolling equivalents of `.xview()`, `.xview_moveto()`, and `.xview_scroll()`.

Next: [8.7. Canvas arc objects](#)

Contents: [Tkinter 8.5 reference: a GUI for Python](#)

Previous: [8.5. Canvas *tagOrId* arguments](#)

Home: [John Shipman's Home Sweet Homepage](#)

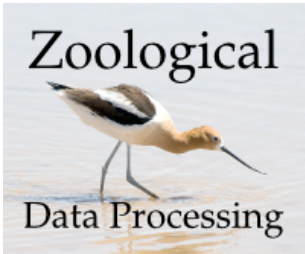
John W. Shipman

Comments welcome: john@nmt.edu

Last updated: 2013-09-09 22:58

URL: <http://www.nmt.edu/~shipman/soft/tkinter/web/canvas-methods.html>

Tkinter 8.5 reference: a GUI for Python



8.7. Canvas arc objects

An *arc object* on a canvas, in its most general form, is a wedge-shaped slice taken out of an ellipse. This includes whole ellipses and circles as special cases. See [Section 8.11, “Canvas oval objects”](#) for more on the geometry of the ellipse drawn.

To create an arc object on a canvas *C*, use:

```
id = C.create_arc(x0, y0, x1, y1, option, ...)
```

The constructor returns the [object ID](#) of the new arc object on canvas *C*.

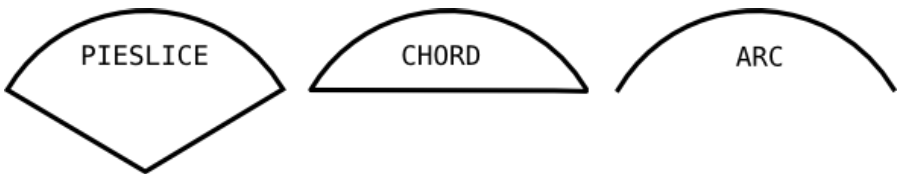
Point (*x0*, *y0*) is the top left corner and (*x1*, *y1*) the lower right corner of a rectangle into which the ellipse is fit. If this rectangle is square, you get a circle.

The various options include:

Table 7. Canvas arc options

activedash	These options apply when the arc is in the tk.ACTIVE state, that is, when the mouse is over the arc. For example, the activefill option specifies the interior color when the arc is active. For option values, see dash, fill, outline, outlinestipple, stipple, and width, respectively.
activefill	
activeoutline	
activeoutlinestipple	
activestipple	
activewidth	
dash	Dash pattern for the outline. See Section 5.13, “Dash patterns” .
dashoffset	Dash pattern offset for the outline. See Section 5.13, “Dash patterns” .
disableddash	These options apply when the arc's state is tk.DISABLED.
disabledfill	

disabledoutline	
disabledoutlinestipple	
disabledstipple	
disabledwidth	
extent	Width of the slice in degrees. The slice starts at the angle given by the <i>start</i> option and extends counterclockwise for <i>extent</i> degrees.
fill	By default, the interior of an arc is transparent, and <code>fill=''</code> will select this behavior. You can also set this option to any color and the interior of the arc will be filled with that color.
offset	Stipple pattern offset for the interior of the arc. See Section 5.14, “Matching stipple patterns” .
outline	The color of the border around the outside of the slice. Default is black.
outlineoffset	Stipple pattern offset for the outline. See Section 5.14, “Matching stipple patterns” .
outlinestipple	If the outline option is used, this option specifies a bitmap used to stipple the border. Default is black, and that default can be specified by setting <code>outlinestipple=''</code> .
start	Starting angle for the slice, in degrees, measured from +x direction. If omitted, you get the entire ellipse.
state	This option is <code>tk.NORMAL</code> by default. It may be set to <code>tk.HIDDEN</code> to make the arc invisible or to <code>tk.DISABLED</code> to gray out the arc and make it unresponsive to events.
stipple	A bitmap indicating how the interior fill of the arc will be stippled. Default is <code>stipple=''</code> (solid). You'll probably want something like <code>stipple='gray25'</code> . Has no effect unless <i>fill</i> has been set to some color.
style	The default is to draw the whole arc; use <code>style=tk.PIESLICE</code> for this style. To draw only the circular arc at the edge of the slice, use <code>style=tk.ARC</code> . To draw the circular arc and the chord (a straight line connecting the endpoints of the arc), use <code>style=tk.CHORD</code> .

	
tags	If a single string, the arc is tagged with that string. Use a tuple of strings to tag the arc with multiple tags. See Section 8.4, "Canvas tags" .
width	Width of the border around the outside of the arc. Default is 1 pixel.

Next: [8.8. Canvas bitmap objects](#)

Contents: [Tkinter 8.5 reference: a GUI for Python](#)

Previous: [8.6. Methods on Canvas widgets](#)

Home: [John Shipman's Home Sweet Homepage](#)

John W. Shipman

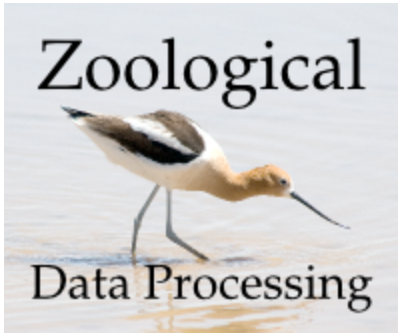
Comments welcome: john@nmt.edu

Last updated: 2013-09-09 22:58

URL: http://www.nmt.edu/~shipman/soft/tkinter/web/create_arc.html

[Next](#) / [Previous](#) / [Contents](#) / [Shipman's homepage](#)

Tkinter 8.5 reference: a GUI for Python



8.8. Canvas bitmap objects

A bitmap object on a canvas is shown as two colors, the background color (for 0 data values) and the foreground color (for 1 values).

To create a bitmap object on a canvas *C*, use:

```
id = C.create_bitmap(x, y, *options ...)
```

which returns the integer ID number of the image object for that canvas.

The *x* and *y* values are the reference point that specifies where the bitmap is placed.

Options include:

Table 8. Canvas bitmap options

activebackground	These options specify the background, bitmap, and foreground values when the bitmap is active, that is, when the mouse is over the bitmap.
activebitmap	
activeforeground	

anchor	The bitmap is positioned relative to point (x, y) . The default is <code>anchor=tk.CENTER</code> , meaning that the bitmap is centered on the (x, y) position. See Section 5.5, “Anchors” for the various anchor option values. For example, if you specify <code>anchor=tk.NE</code> , the bitmap will be positioned so that point (x, y) is located at the northeast (top right) corner of the bitmap.
background	The color that will appear where there are 0 values in the bitmap. The default is <code>background=' '</code> , meaning transparent.
bitmap	The bitmap to be displayed; see Section 5.7, “Bitmaps” .
disabledbackground	These options specify the background, bitmap, and foreground to be used when the bitmap's state is <code>tk.DISABLED</code> .
disabledbitmap	
disabledforeground	
foreground	The color that will appear where there are 1 values in the bitmap. The default is <code>foreground='black'</code> .
state	By default, items are created with <code>state=tk.NORMAL</code> . Use <code>tk.DISABLED</code> to make the item grayed out and unresponsive to events; use <code>tk.HIDDEN</code> to make the item invisible.
tags	If a single string, the bitmap is tagged with that string. Use a tuple of strings to tag the bitmap with multiple tags. See Section 8.4, “Canvas tags” .

Next: [8.9. Canvas image objects](#)

Contents: [Tkinter 8.5 reference: a GUI for Python](#)

Previous: [8.7. Canvas arc objects](#)

Home: [John Shipman's Home Sweet Homepage](#)

John W. Shipman

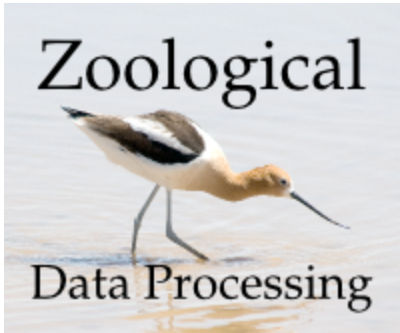
Comments welcome: john@nmt.edu

Last updated: 2013-09-09 22:58

URL: http://www.nmt.edu/~shipman/soft/tkinter/web/create_bitmap.html

[Next](#) / [Previous](#) / [Contents](#) / [Shipman's homepage](#)

Tkinter 8.5 reference: a GUI for Python



8.9. Canvas image objects

To display a graphics image on a canvas *C*, use:

```
id = C.create_image(x, y, option, ...)
```

This constructor returns the integer ID number of the image object for that canvas.

The image is positioned relative to point (x, y) . Options include:

Table 9. Canvas image options

activeimage	Image to be displayed when the mouse is over the item. For option values, see image below.
anchor	The default is anchor=tk.CENTER, meaning that the image is centered on the (x, y) position. See Section 5.5, “Anchors” for the possible values of this option. For example, if you specify anchor=tk.S, the image will be positioned so that point (x, y) is located at the center of the bottom (south) edge of the

	image.
disabledimage	Image to be displayed when the item is inactive. For option values, see image below.
image	The image to be displayed. See Section 5.9, "Images" , above, for information about how to create images that can be loaded onto canvases.
state	Normally, image objects are created in state tk.NORMAL. Set this value to tk.DISABLED to make it grayed-out and unresponsive to the mouse. If you set it to tk.HIDDEN, the item is invisible.
tags	If a single string, the image is tagged with that string. Use a tuple of strings to tag the image with multiple tags. See Section 8.4, "Canvas tags" .

Next: [8.10. Canvas line objects](#)

Contents: [Tkinter 8.5 reference: a GUI for Python](#)

Previous: [8.8. Canvas bitmap objects](#)

Home: [John Shipman's Home Sweet Homepage](#)

John W. Shipman

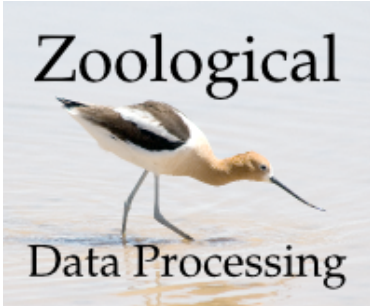
Comments welcome: john@nmt.edu

Last updated: 2013-09-09 22:58

URL: http://www.nmt.edu/~shipman/soft/tkinter/web/create_image.html

[Next](#) / [Previous](#) / [Contents](#) / [Shipman's homepage](#)

Tkinter 8.5 reference: a GUI for Python



8.10. Canvas line objects

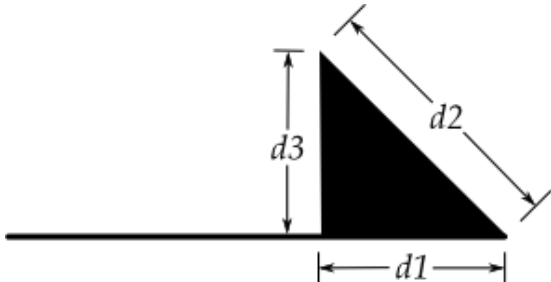
In general, a line can consist of any number of segments connected end to end, and each segment can be straight or curved. To create a canvas line object on a canvas *C*, use:

```
id = C.create_line(x0, y0, x1, y1, ..., xn, yn, option, ...)
```

The line goes through the series of points (x_0, y_0) , (x_1, y_1) , ... (x_n, y_n) . Options include:

Table 10. Canvas line options

activedash	These options specify the dash, fill, stipple, and width values to be used when the line is active, that is, when the mouse is over it.
activefill	
activestipple	
activewidth	
arrow	The default is for the line to have no arrowheads. Use <code>arrow=tk.FIRST</code> to get an arrowhead at the (x_0, y_0) end of the line. Use <code>arrow=tk.LAST</code> to get an arrowhead at the far end. Use <code>arrow=tk.BOTH</code> for arrowheads at both ends.
arrowshape	A tuple $(d1, d2, d3)$ that describes the shape of

	<p>the arrowheads added by the arrow option. Default is (8,10,3).</p> 
capstyle	You can specify the shape of the ends of the line with this option; see Section 5.12, “Cap and join styles” . The default option is tk.BUTT.
dash	To produce a dashed line, specify this option; see Section 5.13, “Dash patterns” . The default appearance is a solid line.
dashoffset	If you specify a dash pattern, the default is to start the specified pattern at the beginning of the line. The dashoffset option allows you to specify that the start of the dash pattern occurs at a given distance after the start of the line. See Section 5.13, “Dash patterns” .
disableddash	The dash, fill, stipple, and width values to be used when the item is in the tk.DISABLED state.
disabledfill	
disabledstipple	
disabledwidth	
fill	The color to use in drawing the line. Default is fill='black'.
joinstyle	For lines that are made up of more than one line segment, this option controls the appearance of the junction between segments. For more details, see Section 5.12, “Cap and join styles” . The default style is ROUND

offset	For stippled lines, the purpose of this option is to match the item's stippling pattern with those of adjacent objects. See Section 5.14, “Matching stipple patterns” ..
smooth	If true, the line is drawn as a series of parabolic splines fitting the point set. Default is false, which renders the line as a set of straight segments.
splinsteps	If the smooth option is true, each spline is rendered as a number of straight line segments. The splinsteps option specifies the number of segments used to approximate each section of the line; the default is splinsteps=12.
state	Normally, line items are created in state <code>tk.NORMAL</code> . Set this option to <code>tk.HIDDEN</code> to make the line invisible; set it to <code>tk.DISABLED</code> to make it unresponsive to the mouse.
stipple	To draw a stippled line, set this option to a bitmap that specifies the stippling pattern, such as <code>stipple='gray25'</code> . See Section 5.7, “Bitmaps” for the possible values.
tags	If a single string, the line is tagged with that string. Use a tuple of strings to tag the line with multiple tags. See Section 8.4, “Canvas tags” .
width	The line's width. Default is 1 pixel. See Section 5.1, “Dimensions” for possible values.

Next: [8.11. Canvas oval objects](#)

Contents: [Tkinter 8.5 reference: a GUI for Python](#)

Previous: [8.9. Canvas image objects](#)

Home: [John Shipman's Home Sweet Homepage](#)

John W. Shipman

Comments welcome: john@nmt.edu

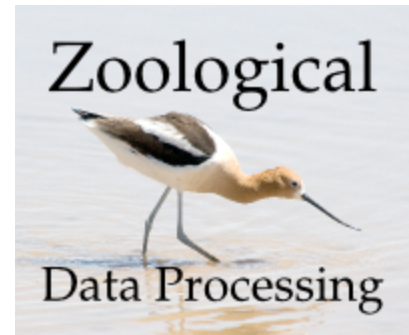
Last updated: 2013-09-09 22:58

URL: http://www.nmt.edu/~shipman/soft/tkinter/web/create_line.html

[Next](#) / [Previous](#) / [Contents](#) / [Shipman's homepage](#)

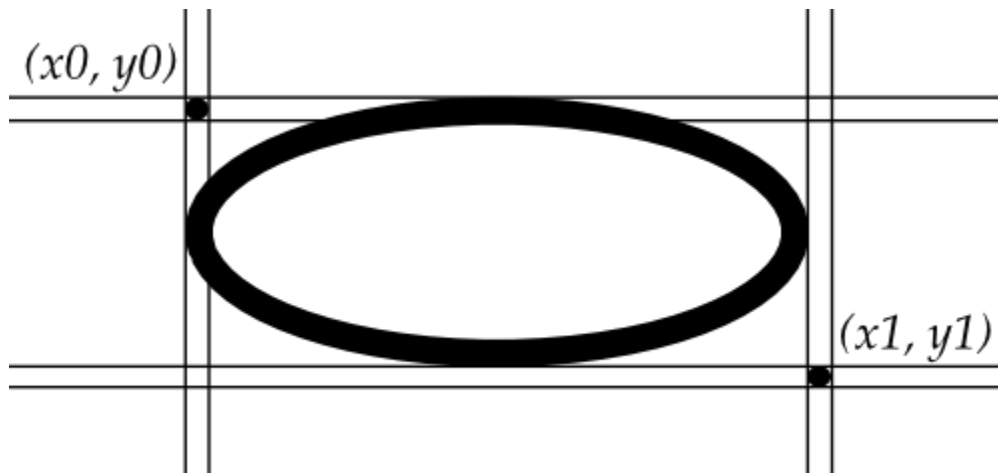
***Tkinter* 8.5**

reference: a GUI for Python



8.11. Canvas oval objects

Ovals, mathematically, are ellipses, including circles as a special case. The ellipse is fit into a rectangle defined by the coordinates (x_0, y_0) of the top left corner and the coordinates (x_1, y_1) of a point just *outside* of the bottom right corner.



The oval will coincide with the top and left-hand lines of this box, but will fit just inside the bottom and right-hand sides.

To create an ellipse on a canvas C , use:

```
id = C.create_oval(x0, y0, x1, y1, option, ...)
```

which returns the [object ID](#) of the new oval object on canvas C .

Options for ovals:

Table 11. Canvas oval options

activedash	These options specify the dash pattern, fill color, outline color, outline stipple pattern, interior stipple pattern, and outline width values to be used when the oval is in the tk.ACTIVE state, that is, when the mouse is over the oval. For option values, see dash, fill, outline, outlinestipple, stipple, and width.
activefill	
activeoutline	
activeoutlinestipple	
activestipple	
activewidth	
dash	To produce a dashed border around the oval, set this option to a dash pattern; see Section 5.13, “Dash patterns”
dashoffset	When using the dash option, the dashoffset option is used to change the alignment of the border's dash pattern relative to the oval. See Section 5.14, “Matching stipple patterns” .
disableddash	These options specify the appearance of the oval when the item's state is tk.DISABLED.
disabledfill	
disabledoutline	
disabledoutlinestipple	
disabledstipple	
disabledwidth	

fill	The default appearance of an oval's interior is transparent, and a value of <code>fill=''</code> will select this behavior. You can also set this option to any color and the interior of the ellipse will be filled with that color; see Section 5.3, “Colors” .
offset	Stipple pattern offset of the interior. See Section 5.14, “Matching stipple patterns” .
outline	The color of the border around the outside of the ellipse. Default is <code>outline='black'</code> .
outlineoffset	Stipple pattern offset of the border. See Section 5.14, “Matching stipple patterns” .
stipple	A bitmap indicating how the interior of the ellipse will be stippled. Default is <code>stipple=''</code> , which means a solid color. A typical value would be <code>stipple='gray25'</code> . Has no effect unless the fill has been set to some color. See Section 5.7, “Bitmaps” .
outlinestipple	Stipple pattern to be used for the border. For option values, see stipple below.
state	By default, oval items are created in state <code>tk.NORMAL</code> . Set this option to <code>tk.DISABLED</code> to make the oval unresponsive to mouse actions. Set it to <code>tk.HIDDEN</code> to make the item

	invisible.
tags	If a single string, the oval is tagged with that string. Use a tuple of strings to tag the oval with multiple tags. See Section 8.4, "Canvas tags" .
width	Width of the border around the outside of the ellipse. Default is 1 pixel; see Section 5.1, "Dimensions" for possible values. If you set this to zero, the border will not appear. If you set this to zero and make the fill transparent, you can make the entire oval disappear.

Next: [8.12. Canvas polygon objects](#)

Contents: [Tkinter 8.5 reference: a GUI for Python](#)

Previous: [8.10. Canvas line objects](#)

Home: [John Shipman's Home Sweet Homepage](#)

John W. Shipman

Comments welcome: john@nmt.edu

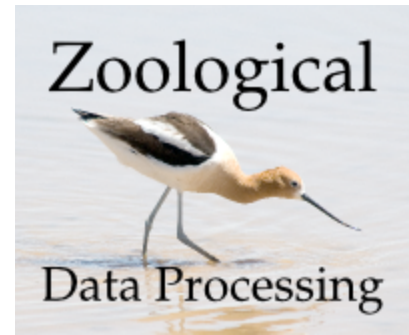
Last updated: 2013-09-09 22:58

URL: http://www.nmt.edu/~shipman/soft/tkinter/web/create_oval.html

[Next](#) / [Previous](#) / [Contents](#) / [Shipman's homepage](#)

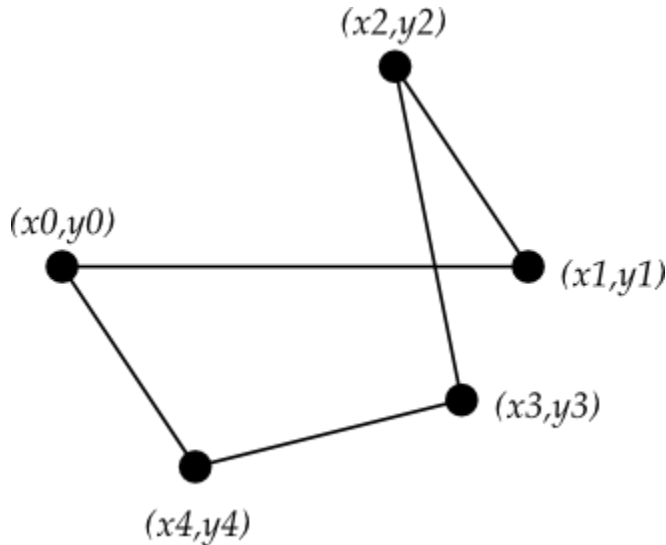
***Tkinter* 8.5**

reference: a GUI for Python



8.12. Canvas polygon objects

As displayed, a polygon has two parts: its outline and its interior. Its geometry is specified as a series of vertices $[(x_0, y_0), (x_1, y_1), \dots, (x_n, y_n)]$, but the actual perimeter includes one more segment from (x_n, y_n) back to (x_0, y_0) . In this example, there are five vertices:



To create a new polygon object on a canvas C :

```
id = C.create_polygon(x0, y0, x1, y1, ..., option, ...)
```

The constructor returns the [object ID](#) for that object. Options:

Table 12. Canvas polygon options

activedash	These options specify the appearance of the polygon when it is in the tk.ACTIVE state, that is, when the mouse is over it. For option values, see dash, fill, outline, outlinestipple, stipple, and width.
activefill	
activeoutline	
activeoutlinestipple	
activestipple	
activewidth	
dash	Use this option to produce a dashed border around the polygon. See Section 5.13, “Dash patterns” .
dashoffset	Use this option to start the dash pattern at some point in its cycle other than the beginning. See Section 5.13, “Dash patterns” .
disableddash	These options specify the appearance of the polygon when its state is tk.DISABLED.
disabledfill	
disabledoutline	
disabledoutlinestipple	
disabledstipple	
disabledwidth	
fill	You can color the interior by setting this option to a color. The default appearance for the interior of a polygon is transparent, and you can set fill='' to get this behavior. See Section 5.3, “Colors” .

joinstyle	This option controls the appearance of the intersections between adjacent sides of the polygon. See Section 5.12, “Cap and join styles” .
offset	Offset of the stipple pattern in the interior of the polygon. See Section 5.14, “Matching stipple patterns” .
outline	Color of the outline; defaults to outline='', which makes the outline transparent.
outlineoffset	Stipple offset for the border. See Section 5.14, “Matching stipple patterns” .
outlinestipple	Use this option to get a stippled border around the polygon. The option value must be a bitmap; see Section 5.7, “Bitmaps” .
smooth	The default outline uses straight lines to connect the vertices; use smooth=0 to get that behavior. If you use smooth=1, you get a continuous spline curve. Moreover, if you set smooth=1, you can make any segment straight by duplicating the coordinates at each end of that segment.
splinsteps	If the smooth option is true, each spline is rendered as a number of straight line segments. The splinsteps option specifies the number of segments used to

	approximate each section of the line; the default is <code>splinsteps=12</code> .
<code>state</code>	By default, polygons are created in the <code>tk.NORMAL</code> state. Set this option to <code>tk.HIDDEN</code> to make the polygon invisible, or set it to <code>tk.DISABLED</code> to make it unresponsive to the mouse.
<code>stipple</code>	A bitmap indicating how the interior of the polygon will be stippled. Default is <code>stipple=''</code> , which means a solid color. A typical value would be <code>stipple='gray25'</code> . Has no effect unless the fill has been set to some color. See Section 5.7, “Bitmaps” .
<code>tags</code>	If a single string, the polygon is tagged with that string. Use a tuple of strings to tag the polygon with multiple tags. See Section 8.4, “Canvas tags” .
<code>width</code>	Width of the outline; defaults to 1. See Section 5.1, “Dimensions” .

Next: [8.13. Canvas rectangle objects](#)

Contents: [Tkinter 8.5 reference: a GUI for Python](#)

Previous: [8.11. Canvas oval objects](#)

Home: [John Shipman's Home Sweet Homepage](#)

John W. Shipman

Comments welcome: john@nmt.edu

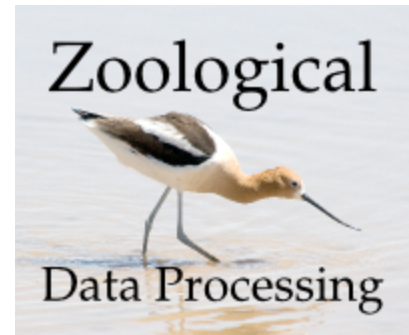
Last updated: 2013-09-09 22:58

URL: http://www.nmt.edu/~shipman/soft/tkinter/web/create_polygon.html

[Next](#) / [Previous](#) / [Contents](#) / [Shipman's homepage](#)

***Tkinter* 8.5**

reference: a GUI for Python



8.13. Canvas rectangle objects

Each rectangle is specified as two points: (x_0, y_0) is the top left corner, and (x_1, y_1) is the location of the pixel just *outside* of the bottom right corner.

For example, the rectangle specified by top left corner $(100, 100)$ and bottom right corner $(102, 102)$ is a square two pixels by two pixels, including pixel $(101, 101)$ but *not* including $(102, 102)$.

Rectangles are drawn in two parts:

- The outline lies inside the rectangle on its top and left sides, but *outside* the rectangle on its bottom and right side. The default appearance is a one-pixel-wide black border.

For example, consider a rectangle with top left corner $(10, 10)$ and bottom right corner $(11, 11)$. If you request no border (`width=0`) and green fill (`fill='green'`), you will get one green pixel at $(10, 10)$. However, if you request the same options with a black border (`width=1`), you will get four black pixels at $(10, 10)$, $(10, 11)$, $(11, 10)$, and $(11, 11)$.

- The fill is the area inside the outline. Its default

appearance is transparent.

To create a rectangle object on canvas *C*:

```
id = C.create_rectangle(x0, y0, x1, y1, option, ...)
```

This constructor returns the [object ID](#) of the rectangle on that canvas. Options include:

Table 13. Canvas rectangle options

activedash	These options specify the appearance of the rectangle when its state is tk.ACTIVE, that is, when the mouse is on top of the rectangle. For option values, refer to dash, fill, outline, outlinestipple, stipple, and width below.
activefill	
activeoutline	
activeoutlinestipple	
activestipple	
activewidth	
dash	To produce a dashed border around the rectangle, use this option to specify a dash pattern. See Section 5.13, “Dash patterns” .
dashoffset	Use this option to start the border's dash pattern at a different point in the cycle; see Section 5.13, “Dash patterns” .
disableddash	These options specify the appearance of the rectangle when its state is tk.DISABLED.
disabledfill	
disabledoutline	
disabledoutlinestipple	

<code>disabledstipple</code>	
<code>disabledwidth</code>	
<code>fill</code>	By default, the interior of a rectangle is empty, and you can get this behavior with <code>fill=''</code> . You can also set the option to a color; see Section 5.3, “Colors” .
<code>offset</code>	Use this option to change the offset of the interior stipple pattern. See Section 5.14, “Matching stipple patterns” .
<code>outline</code>	The color of the border. Default is <code>outline='black'</code> .
<code>outlineoffset</code>	Use this option to adjust the offset of the stipple pattern in the outline; see Section 5.14, “Matching stipple patterns” .
<code>outlinestipple</code>	Use this option to produce a stippled outline. The pattern is specified by a bitmap; see Section 5.7, “Bitmaps” .
<code>state</code>	By default, rectangles are created in the <code>tk.NORMAL</code> state. The state is <code>tk.ACTIVE</code> when the mouse is over the rectangle. Set this option to <code>tk.DISABLED</code> to gray out the rectangle and make it unresponsive to mouse events.
<code>stipple</code>	A bitmap indicating how the interior of the rectangle will be stippled. Default is <code>stipple=''</code> , which means

	a solid color. A typical value would be <code>stipple='gray25'</code> . Has no effect unless the fill has been set to some color. See Section 5.7, “Bitmaps” .
tags	If a single string, the rectangle is tagged with that string. Use a tuple of strings to tag the rectangle with multiple tags. See Section 8.4, “Canvas tags” .
width	Width of the border. Default is 1 pixel. Use <code>width=0</code> to make the border invisible. See Section 5.1, “Dimensions” .

Next: [8.14. Canvas text objects](#)

Contents: [Tkinter 8.5 reference: a GUI for Python](#)

Previous: [8.12. Canvas polygon objects](#)

Home: [John Shipman's Home Sweet Homepage](#)

John W. Shipman

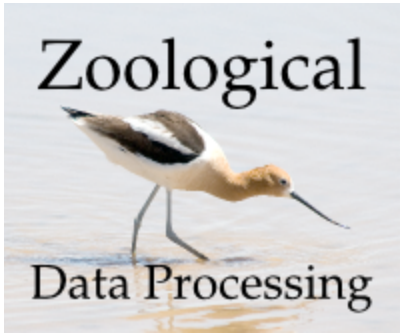
Comments welcome: john@nmt.edu

Last updated: 2013-09-09 22:58

URL: http://www.nmt.edu/~shipman/soft/tkinter/web/create_rectangle.html

[Next](#) / [Previous](#) / [Contents](#) / [Shipman's homepage](#)

Tkinter 8.5 reference: a GUI for Python



8.14. Canvas text objects

You can display one or more lines of text on a canvas *C* by creating a text object:

```
id = C.create_text(x, y, option, ...)
```

This returns the [object ID](#) of the text object on canvas *C*. Options include:

Table 14. Canvas text options

activefill	The text color to be used when the text is active, that is, when the mouse is over it. For option values, see fill below.
activestipple	The stipple pattern to be used when the text is active. For option values, see stipple below.
anchor	The default is anchor=tk.CENTER, meaning that the text is centered vertically and horizontally around position (x, y). See Section 5.5, “Anchors” for possible values. For example, if you specify anchor=tk.SW,

	the text will be positioned so its lower left corner is at point (x, y).
disabledfill	The text color to be used when the text object's state is tk.DISABLED. For option values, see fill below.
disabledstipple	The stipple pattern to be used when the text is disabled. For option values, see stipple below.
fill	The default text color is black, but you can render it in any color by setting the fill option to that color. See Section 5.3, "Colors" .
font	If you don't like the default font, set this option to any font value. See Section 5.4, "Type fonts" .
justify	For multi-line textual displays, this option controls how the lines are justified: tk.LEFT (the default), tk.CENTER, or tk.RIGHT.
offset	The stipple offset to be used in rendering the text. For more information, see Section 5.14, "Matching stipple patterns" .
state	By default, the text item's state is tk.NORMAL. Set this option to tk.DISABLED to make it unresponsive to mouse events, or set it to tk.HIDDEN to make it invisible.
stipple	A bitmap indicating how the text will be stippled. Default is stipple='', which means solid. A typical value would be stipple='gray25'. See Section 5.7, "Bitmaps" .

tags	If a single string, the text object is tagged with that string. Use a tuple of strings to tag the object with multiple tags. See Section 8.4, “Canvas tags” .
text	The text to be displayed in the object, as a string. Use newline characters ('\n') to force line breaks.
width	If you don't specify a width option, the text will be set inside a rectangle as long as the longest line. However, you can also set the width option to a dimension, and each line of the text will be broken into shorter lines, if necessary, or even broken within words, to fit within the specified width. See Section 5.1, “Dimensions” .

You can change the text displayed in a text item.

- To retrieve the text from an item with [object ID](#) *I* on a canvas *C*, call *C.itemcget(I, 'text')*.
- To replace the text in an item with [object ID](#) *I* on a canvas *C* with the text from a string *S*, call *C.itemconfigure(I, text=S)*.

A number of canvas methods allow you to manipulate text items. See [Section 8.6, “Methods on Canvas widgets”](#), especially *dchars*, *focus*, *icursor*, *index*, and *insert*.

Next: [8.15. Canvas window objects](#)

Contents: [Tkinter 8.5 reference: a GUI for Python](#)

Previous: [8.13. Canvas rectangle objects](#)

Home: [John Shipman's Home Sweet Homepage](#)

John W. Shipman

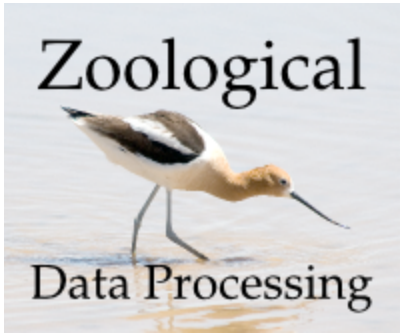
Comments welcome: john@nmt.edu

Last updated: 2013-09-09 22:58

URL: http://www.nmt.edu/~shipman/soft/tkinter/web/create_text.html

[Next](#) / [Previous](#) / [Contents](#) / [Shipman's homepage](#)

Tkinter 8.5 reference: a GUI for Python



8.15. Canvas window objects

You can place any *Tkinter* widget onto a canvas by using a *canvas window* object. A window is a rectangular area that can hold one *Tkinter* widget. The widget must be the child of the same top-level window as the canvas, or the child of some widget located in the same top-level window.

If you want to put complex multi-widget objects on a canvas, you can use this method to place a *Frame* widget on the canvas, and then place other widgets inside that frame.

To create a new canvas window object on a canvas *C*:

```
id = C.create_window(x, y, option, ...)
```

This returns the [object ID](#) for the window object. Options include:

Table 15. Canvas window options

anchor	The default is anchor=tk.CENTER, meaning that the window is centered on the (x, y) position. See Section 5.5, “Anchors” for the possible values. For example, if you specify anchor=tk.E, the window will
--------	---

	be positioned so that point (x, y) is on the midpoint of its right-hand (east) edge.
height	The height of the area reserved for the window. If omitted, the window will be sized to fit the height of the contained widget. See Section 5.1, “Dimensions” for possible values.
state	By default, window items are in the <code>tk.NORMAL</code> state. Set this option to <code>tk.DISABLED</code> to make the window unresponsive to mouse input, or to <code>tk.HIDDEN</code> to make it invisible.
tags	If a single string, the window is tagged with that string. Use a tuple of strings to tag the window with multiple tags. See Section 8.4, “Canvas tags” .
width	The width of the area reserved for the window. If omitted, the window will be sized to fit the width of the contained widget.
window	Use <code>window=w</code> where w is the widget you want to place onto the canvas. If this is omitted initially, you can later call <code>C.itemconfigure (id, window=w)</code> to place the widget w onto the canvas, where id is the window's object ID..

Next: [9. The Checkbutton widget](#)

Contents: [Tkinter 8.5 reference: a GUI for Python](#)

Previous: [8.14. Canvas text objects](#)

Home: [John Shipman's Home Sweet Homepage](#)

John W. Shipman

Comments welcome: john@nmt.edu

Last updated: 2013-09-09 22:58

URL: http://www.nmt.edu/~shipman/soft/tkinter/web/create_window.html