The tkinter Canvas Widget

The **Canvas** widget provides structured graphics facilities for tkinter. This is a highly versatile widget which can be used to draw graphs and plots, create graphics editors, and implement various kinds of custom widgets.

When to use the Canvas Widget

The canvas is a general purpose widget, which is typically used to display and edit graphs and other drawings.

Another common use for this widget is to implement various kinds of custom widgets. For example, you can use a canvas as a completion bar, by drawing and updating a rectangle on the canvas.

Patterns

To draw things in the canvas, use the **create** methods to add new items.

```
from tkinter import *

master = Tk()

w = Canvas(master, width=200, height=100)
w.pack()

w.create_line(0, 0, 200, 100)
w.create_line(0, 100, 200, 0, fill="red", dash=(4, 4))

w.create_rectangle(50, 25, 150, 75, fill="blue")

mainloop()
```

Note that items added to the canvas are kept until you remove them. If you want to change the drawing, you can either use methods like **coords**, **itemconfig**, and **move** to modify the items, or use **delete** to remove them.

```
i = w.create_line(xy, fill="red")

w.coords(i, new_xy) # change coordinates
w.itemconfig(i, fill="blue") # change color

w.delete(i) # remove

w.delete(ALL) # remove all items
```

Concepts

To display things on the canvas, you create one or more *canvas items*, which are placed in a stack. By default, new items are drawn on top of items already on the canvas.

tkinter provides lots of methods allowing you to manipulate the items in various ways. Among other things, you can attach (*bind*) event callbacks to individual canvas items.

Canvas Items

The **Canvas** widget supports the following standard items:

- **arc** (arc, chord, or pieslice)
- **bitmap** (built-in or read from XBM file)
- **image** (a BitmapImage or PhotoImage instance)
- **line**
- **oval** (a circle or an ellipse)
- **polygon**
- **rectangle**
- **text**
- **window**

Chords, pieslices, ovals, polygons, and rectangles consist of both an outline and an interior area, either of which can be made transparent (and if you insist, you can make both transparent).

Window items are used to place other tkinter widgets on top of the canvas; for these items, the Canvas widget simply acts like a geometry manager.

You can also write your own item types in C or C++ and plug them into tkinter via Python extension modules.

Coordinate Systems

The **Canvas** widget uses two coordinate systems; the window coordinate system (with (0, 0) in the upper left corner), and a canvas coordinate system which specify where the items are drawn. By scrolling the canvas, you can specify which part of the canvas coordinate system to show in the window.

The **scrollregion** option is used to limit scrolling operations for the canvas. To set this, you can usually use something like:

```
canvas.config(scrollregion=canvas.bbox(ALL))
```

To convert from window coordinates to canvas coordinates, use the **canvasx** and **canvasy** methods:

```
def callback(event):
    canvas = event.widget
    x = canvas.canvasx(event.x)
    y = canvas.canvasy(event.y)
    print canvas.find_closest(x, y)
```

Item Specifiers: Handles and Tags

The **Canvas** widget allows you to identify items in several ways. Everywhere a method expects an item specifier, you can use one of the following:

- item handles (integers)
- tags
- **ALL**
- **CURRENT**

Item handles are integer values used to identify a specific item on the canvas. tkinter automatically assigns a new handle to each new item created on the canvas. Item handles can be passed to the various canvas methods either as integers or as strings.

Tags are symbolic names attached to items. Tags are ordinary strings, and they can contain anything except whitespace (as long as they don't look like item handles).

An item can have zero or more tags associated with it, and the same tag can be used for more than one item. However, unlike the **Text** widget, the **Canvas** widget doesn't allow you to create bindings or otherwise configure tags for which there are no existing items. Tags are owned by the items, not the widget itself. All such operations are ignored.

You can either specify the tags via an option when you create the item, set them via the **itemconfig** method, or add them using the **addtag_withtag** method. The **tags** option takes either a single tag string, or a tuple of strings.

```
item = canvas.create_line(0, 0, 100, 100, tags="uno")
canvas.itemconfig(item, tags=("one", "two"))
canvas.addtag_withtag("three", "one")
```

To get all tags associated with a specific item, use **gettags**. To get the handles for all items having a given tag, use **find_withtag**.

```
>>> print canvas.gettags(item)
('one', 'two', 'three')
>>> print canvas.find_withtag("one")
(1,)
```

The **Canvas** widget also provides two predefined tags:

ALL (or the string "all") matches all items on the canvas.

CURRENT (or “current”) matches the item under the mouse pointer, if any. This can be used inside mouse event bindings to refer to the item that triggered the callback.

Printing

The tkinter widget supports printing to Postscript printers.

Performance Issues

The **Canvas** widget implements a straight-forward damage/repair display model. Changes to the canvas, and external events such as **Expose**, are all treated as “damage” to the screen. The widget maintains a **dirty rectangle** to keep track of the damaged area.

When the first damage event arrives, the canvas registers an idle task (using **after_idle**) which is used to “repair” the canvas when the program gets back to the tkinter main loop. You can force updates by calling the **update_idletasks** method.

When it’s time to redraw the canvas, the widget starts by allocating a pixmap (on X windows, this is an image memory stored on the display) with the same size as the dirty rectangle.

It then loops over the canvas items, and redraws *all* items for which the bounding box touch the dirty rectangle (this means that diagonal lines may be redrawn also if they don’t actually cover the rectangle, but this is usually no big deal).

Finally, the widget copies the pixmap to the display, and releases the pixmap. The copy operation is a very fast operation on most modern hardware.

Since the canvas uses a *single* dirty rectangle, you can sometimes get better performance by forcing updates. For example, if you’re changing things in different parts of the canvas without returning to the main loop, adding explicit calls to **update_idletasks()** allows the canvas to update a few small rectangles, instead of a large one with many more objects.

Reference

Canvas(master=None, **options) (class) [#]

A structured graphics canvas.

master
: Parent widget.

***options*
: Widget options. See the description of the **config** method for a list of available options.

addtag(tag, method, *args) [#]

Adds a tag to a number of items. Application code should use more specific methods wherever possible (that is, use **addtag_above** instead of **addtag(“above”)**, and so on.

tag
: The tag to add.

method
: How to add a new tag. This can be one of “above”, “all”, “below”, “closest”, “enclosed”, “overlapping” or “withtag”.

**args*
: Additional arguments. For details, see the description of the individual method.

addtag_above(tag, item) [#]

Adds a tag to the item just above the given item.

tag
: The tag to add.

item
: The tag or id of the reference item.

addtag_all(tag) [#]

Adds a tag to all items on the canvas. This is a shortcut for **addtag_withtag(newtag, ALL)**.

tag
: The tag to add.

addtag_below(tag, item) [#]

Adds a tag to the item just below the given item.

tag
: The tag to add.

item
: The tag or id of the reference item.

addtag_closest(tag, x, y, halo=None, start=None) [#]

Adds a tag to the item closest to the given coordinate. Note that the position is given in canvas coordinates, and that this method always succeeds if there's at least one item in the canvas. To add tags to items within a certain distance from the position, use **~~add_overlapping~~** (dead link) with a small rectangle centered on the position.

tag
: The tag to add.

x
: The horizontal coordinate.

y
: The vertical coordinate.

halo
: Optional halo distance.

start
: Optional start item.

addtag_enclosed(tag, x1, y1, x2, y2) [#]

Adds a tag to all items enclosed by the given rectangle.

tag
: The tag to add.

x1
: Left coordinate.

y1
: Top coordinate.

x2
: Right coordinate.

y2
: Bottom coordinate.

addtag_overlapped(tag, x1, y1, x2, y2) [#]

Adds a tag to all items overlapping the given rectangle. This includes items that are completely enclosed by it.

tag
: The tag to add.

x1
: Left coordinate.

y1
: Top coordinate.

x2
: Right coordinate.

y2
: Bottom coordinate.

addtag_withtag(tag, item) [#]

Adds a tag to all items having the given tag.

tag
: The tag to add.

item
: The reference item. If a tag is given, the new tag is added to all items that have this tag. You can also give an id, to add a tag to a single item.

bbox(item=None) [#]

Returns the bounding box for all matching items. If the tag is omitted, the bounding box for all items is returned. Note that the bounding box is approximate and may differ a few pixels from the real value.

item
: Item specifier. If omitted, the bounding box for all elements on the canvas.

Returns:
: The bounding box, as a 4-tuple.

canvasx(x, gridspacing=None) [#]

Converts a window coordinate to a canvas coordinate.

x
: Screen coordinate.

gridspacing
: Optional grid spacing. The coordinate is rounded to the nearest grid coordinate.

Returns:
: Canvas coordinate.

canvasy(y, gridspacing=None) [#]

Converts a window coordinate to a canvas coordinate.

y
: Screen coordinate.

gridspacing
: Optional grid spacing. The coordinate is rounded to the nearest grid coordinate.

Returns:
: Canvas coordinate.

config(options)** [#]

Modifies one or more widget options. If no options are given, the method returns a dictionary containing all current option values.

***options*
: Widget options.

background=
: Canvas background color. Defaults to the standard widget background color. (the database name is background, the class is Background)

bg=
: Same as **background**.

borderwidth=
: Width of the canvas border. The default is 0 (no border). (borderWidth/BorderWidth)

bd=
: Same as **borderwidth**.

closeenough=
: The default value is 1. (closeEnough/CloseEnough)

confine=
: The default value is 1. (confine/Confine)

cursor=
: The cursor to use when the mouse is moved over the canvas. (cursor/Cursor)

height=
Canvas width. Default value is '7c'. (height/Height)
highlightbackground=
The color to use for the highlight border when the canvas does not have focus. The default is system specific. (highlightBackground/HighlightBackground)
highlightcolor=
The color to use for the highlight border when the canvas has focus. The default is system speciific. (highlightColor/HighlightColor)
highlightthickness=
The width of the highlight border. The default is system specific (usually one or two pixels). (highlightThickness/HighlightThickness)
insertbackground=
The color to use for the text insertion cursor. The default is system specific. (insertBackground/Foreground)
insertborderwidth=
Width of the insertion cursor's border. If this is set to a non-zero value, the cursor is drawn using the **RAISED** border style. (insertBorderWidth/BorderWidth)
insertofftime=
Together with **insertontime**, this option controls cursor blinking. Both values are given in milliseconds. (insertOffTime/OffTime)
insertontime=
See **insertofftime**. (insertOnTime/OnTime)
insertwidth=
Width of the insertion cursor. Usually one or two pixels. (insertWidth/InsertWidth)
offset=
Default value is '0,0'. (offset/Offset)
relief=
Border style. The default is **FLAT**. Other possible values are **SUNKEN**, **RAISED**, **GROOVE**, and **RIDGE**. (relief/Relief)
scrollregion=
Canvas scroll region. No default value. (scrollRegion/ScrollRegion)
selectbackground=
Selection background color. The default is system and display specific. (selectBackground/Foreground)
selectborderwidth=
Selection border width. The default is system specific. (selectBorderWidth/BorderWidth)
selectforeground=
Selection text color. The default is system specific. (selectForeground/Background)
state=
Canvas state. One of NORMAL, DISABLED, or HIDDEN. The default is NORMAL. Note that this is a global setting, but individual canvas items can use the item-level **state** option to override this setting. (state/State)
takefocus=
Indicates that the user can use the **Tab** key to move to this widget. Default is an empty string, which means that the canvas widget accepts focus only if it has any keyboard bindings. (takeFocus/TakeFocus)
width=
Canvas width. Default value is '10c'. (width/Width)
xscrollcommand=
Used to connect a canvas to a horizontal scrollbar. This option should be set to the **set** method of the corresponding scrollbar. (xScrollCommand/ScrollCommand)
xscrollincrement=
Default value is 0. (xScrollIncrement/ScrollIncrement)
yscrollcommand=
Used to connect a canvas to a vertical scrollbar. This option should be set to the **set** method of the corresponding scrollbar. (yScrollCommand/ScrollCommand)
yscrollincrement=
Default value is 0. (yScrollIncrement/ScrollIncrement)

coords(item, *coords) [#]

Returns the coordinates for an item.

item
Item specifier (tag or id).

**coords*
: Optional list of coordinate pairs. If given, the coordinates will replace the current coordinates for all matching items.

Returns:
: If no coordinates are given, this method returns the coordinates for the matching item. If the item specifier matches more than one item, the coordinates for the first item found is returned.

create_arc(bbox, **options) [#]

Draws an arc, pieslice, or chord on the canvas. The new item is drawn on top of the existing items.

bbox
: Bounding box for the full arc.

***options*
: Arc options.

activedash=
activefill=
: Fill color to use when the mouse pointer is moved over the item, if different from **fill**.

activeoutline=
activeoutlinestipple=
activestipple=
activewidth=
: Default is 0.0.

dash=
: Outline dash pattern, given as a list of segment lengths. Only the odd segments are drawn.

dashoffset=
: Default is 0.

disableddash=
disabledfill=
: Fill color to use when the item is disabled, if different from **fill**.

disabledoutline=
disabledoutlinestipple=
disabledstipple=
disabledwidth=
: Default is 0.0.

extent=
: The size, relative to the **start** angle. Default is 90.0.

fill=
: Fill color. An empty string means transparent.

offset=
: Default is “0,0”.

outline=
: Outline color. Default is “black”.

outlineoffset=
: Default is “0,0”.

outlinestipple=
: Outline stipple pattern.

start=
: Start angle. Default is 0.0.

state=
: Item state. One of NORMAL, DISABLED, or HIDDEN.

stipple=
: Stipple pattern.

style=
: One of PIESLICE, CHORD, or ARC. Default is PIESLICE.

tags=
: A tag to attach to this item, or a tuple containing multiple tags.

width=
: Default is 1.0.

Returns:
: The item id.

create_bitmap(position, **options) [#]

Draws a bitmap on the canvas.

position
: Bitmap position, given as two coordinates.

***options*
: Bitmap options.

activebackground=

activebitmap=

activeforeground=

anchor=
: Where to place the bitmap relative to the given position. Default is CENTER.

background=
: Background color, used for pixels that are “off”. Use an empty string to make the background transparent. Default is transparent.

bitmap=
: The bitmap descriptor. See **BitmapImage** for more information. (To display a **BitmapImage** object, use the **create_image** function.)

disabledbackground=

disabledbitmap=

disabledforeground=

foreground=
: Foreground colors, used for pixels that are “on”. Default is “black”.

state=
: Item state. One of NORMAL, DISABLED, or HIDDEN.

tags=
: A tag to attach to this item, or a tuple containing multiple tags.

Returns:
: The item id.

create_image(position, **options) [#]

Draws an image on the canvas.

position
: Image position, given as two coordinates.

***options*
: Image options.

activeimage=

anchor=
: Where to place the image relative to the given position. Default is CENTER.

disabledimage=

image=
: The image object. This should be a **PhotoImage** or **BitmapImage**, or a compatible object (such as the PIL PhotoImage). The application must keep a reference to the image object.

state=
: Item state. One of NORMAL, DISABLED, or HIDDEN.

tags=
: A tag to attach to this item, or a tuple containing multiple tags.

Returns:
: The item id.

create_line(coords, **options) [#]

Draws a line on the canvas.

coords
: Image coordinates.

***options*
: Line options.

activedash=

activefill=
: Line color to use when the mouse pointer is moved over the item, if different from **fill**.

activestipple=

activewidth=
: Default is 0.0.

arrow=

Default is NONE.

arrowshape=
: Default is “8 10 3”.

capstyle=
: Default is BUTT.

dash=
: Dash pattern, given as a list of segment lengths. Only the odd segments are drawn.

dashoffset=
: Default is 0.

disableddash=

disabledfill=
: Line color to use when the item is disabled, if different from **fill**.

disabledstipple=

disabledwidth=
: Default is 0.0.

fill=
: Line color. Default is “black”.

joinstyle=
: Default is ROUND.

offset=
: Default is “0,0”.

smooth=
: Default is 0.

splinesteps=
: Default is 12.

state=
: Item state. One of NORMAL, DISABLED, or HIDDEN.

stipple=
: Stipple pattern.

tags=
: A tag to attach to this item, or a tuple containing multiple tags.

width=
: Default is 1.0.

Returns:
: The item id.

create_oval(bbox, **options) [#]

Draws an ellipse on the canvas.

bbox
: Ellipse coordinates.

***options*
: Ellipse options.

activedash=

activefill=
: Fill color to use when the mouse pointer is moved over the item, if different from **fill**.

activeoutline=

activeoutlinestipple=

activestipple=

activewidth=
: Default is 0.0.

dash=
: Outline dash pattern, given as a list of segment lengths. Only the odd segments are drawn.

dashoffset=
: Default is 0.

disableddash=

disabledfill=
: Fill color to use when the item is disabled, if different from **fill**.

disabledoutline=

disabledoutlinestipple=

disabledstipple=

disabledwidth=
: Default is 0.

fill=
: Fill color. An empty string means transparent.

offset=
: Default is “0,0”.

outline=
: Outline color. Default is “black”.

outlineoffset=
: Default is “0,0”.

outlinestipple=
: Outline stipple pattern.

state=
: Item state. One of NORMAL, DISABLED, or HIDDEN.

stipple=
: Stipple pattern.

tags=
: A tag to attach to this item, or a tuple containing multiple tags.

width=
: Default is 1.0.

Returns:
: The item id.

create_polygon(coords, **options) [#]

Draws a polygon on the canvas.

coords
: Polygon coordinates.

***options*
: Polygon options.

activedash=

activefill=
: Fill color to use when the mouse pointer is moved over the item, if different from **fill**.

activeoutline=

activeoutlinestipple=

activestipple=

activewidth=
: Default is 0.0.

dash=
: Outline dash pattern, given as a list of segment lengths. Only the odd segments are drawn.

dashoffset=
: Default is 0.

disableddash=

disabledfill=
: Fill color to use when the item is disabled, if different from **fill**.

disabledoutline=

disabledoutlinestipple=

disabledstipple=

disabledwidth=
: Default is 0.0.

fill=
: Fill color. Default is “black”.

joinstyle=
: Default is ROUND.

offset=
: Default is “0,0”.

outline=
: Outline color.

outlineoffset=
: Default is “0,0”.

outlinestipple=
: Outline stipple pattern.

smooth=
: Default is 0.

splinesteps=
: Default is 12.

state=
: Item state. One of NORMAL, DISABLED, or HIDDEN.

stipple=
: Stipple pattern.

tags=
: A tag to attach to this item, or a tuple containing multiple tags.

width=
: Default is 1.0.

Returns:
: The item id.

create_rectangle(bbox, **options) [#]

Draws a rectangle on the canvas.

bbox
: Rectangle bounding box.

***options*
: Rectangle options.

activedash=

activefill=
: Fill color to use when the mouse pointer is moved over the item, if different from **fill**.

activeoutline=

activeoutlinestipple=

activestipple=

activewidth=
: Default is 0.0.

dash=
: Outline dash pattern, given as a list of segment lengths. Only the odd segments are drawn.

dashoffset=
: Default is 0.

disableddash=

disabledfill=
: Fill color to use when the item is disabled, if different from **fill**.

disabledoutline=

disabledoutlinestipple=

disabledstipple=

disabledwidth=
: Default is 0.

fill=
: Fill color. An empty string means transparent.

offset=
: Default is “0,0”.

outline=
: Outline color. Default is “black”.

outlineoffset=
: Default is “0,0”.

outlinestipple=
: Outline stipple pattern.

state=
: Item state. One of NORMAL, DISABLED, or HIDDEN.

stipple=
: Stipple pattern.

tags=
: A tag to attach to this item, or a tuple containing multiple tags.

width=
: Default is 1.0.

Returns:
: The item id.

create_text(position, **options) [#]

Draws text on the canvas.

position
: Text position, given as two coordinates. By default, the text is centered on this position. You can override this with the **anchor** option. For example, if the coordinate is the upper left corner, set the **anchor** to **NW**.

***options*
: Text options.

activefill=
: Text color to use when the mouse pointer is moved over the item, if different from **fill**.

activestipple=

anchor=
: Where to place the text relative to the given position. Default is CENTER.

disabledfill=
: Text color to use when the item is disabled, if different from **fill**.

disabledstipple=

fill=
: Text color. Default is “black”.

font=
: Font specifier. Default is system specific.

justify=
: Default is LEFT.

offset=
: Default is “0,0”.

state=
: Item state. One of NORMAL, DISABLED, or HIDDEN.

stipple=
: Stipple pattern.

tags=
: A tag to attach to this item, or a tuple containing multiple tags.

text=
: The text to display.

width=
: Maximum line length. Lines longer than this value are wrapped. Default is 0 (no wrapping).

Returns:
: The item id.

create_window(position, **options) [#]

Places a tkinter widget on the canvas. Note that widgets are drawn on top of the canvas (that is, the canvas acts like a geometry manager). You cannot draw other canvas items on top of a widget.

position
: Window position, given as two coordinates.

***options*
: Window options.

anchor=
: Where to place the widget relative to the given position. Default is CENTER.

height=
: Window height. Default is to use the window’s requested height.

state=
: Item state. One of NORMAL, DISABLED, or HIDDEN.

tags=
: A tag to attach to this item, or a tuple containing multiple tags.

width=
: Window width. Default is to use the window’s requested width.

window=
: Window object.

Returns:
: The item id.

dchars(item, from, to=None) [#]

Deletes text from an editable item.

item
: Item specifier.

from
: Where to start deleting text.

to
: Where to stop deleting text. If omitted, a single character is removed.

delete(item) [#]

Deletes all matching items. It is not an error to give an item specifier that doesn't match any items.

item
: Item specifier (tag or id).

dtag(item, tag=None) [#]

Removes the given tag from all matching items. If the tag is omitted, all tags are removed from the matching items. It is not an error to give a specifier that doesn't match any items.

item
: The item specifier (tag or id).

tag
: The tag to remove from matching items. If omitted, all tags are removed.

find_above(item) [#]

Returns the item just above the given item.

item
: Reference item.

find_all() [#]

Returns all items on the canvas. This method returns a tuple containing the identities of all items on the canvas, with the topmost item last (that is, if you haven't change the order using **lift** or **lower**, the items are returned in the order you created them). This is shortcut for **find_withtag(ALL)**.

Returns:
: A tuple containing all items on the canvas.

find_below(item) [#]

Returns the item just below the given item.

item
: Reference item.

find_closest(x, y, halo=None, start=None) [#]

Returns the item closest to the given position. Note that the position is given in canvas coordinates, and that this method always succeeds if there's at least one item in the canvas. To find items within a certain distance from a position, use **find_overlapping** with a small rectangle centered on the position.

x
: Horizontal screen coordinate.

y
: Vertical screen coordinate.

halo
: Optional halo distance.

start
: Optional start item.

Returns:
: An item specifier.

find_enclosed(x1, y1, x2, y2) [#]

Finds all items completely enclosed by the rectangle (x1, y1, x2, y2).

x1

Left edge.

y1
: Upper edge.

x2
: Right edge.

y2
: Lower edge.

Returns:
: A tuple containing all matching items.

find_overlapping(x1, y1, x2, y2) [#]

Finds all items that overlap the given rectangle, or that are completely enclosed by it.

x1
: Left edge.

y1
: Upper edge.

x2
: Right edge.

y2
: Lower edge.

Returns:
: A tuple containing all matching items.

find_withtag(item) [#]

Finds all items having the given specifier.

item
: Item specifier.

focus(item=None) [#]

Moves focus to the given item. If the item has keyboard bindings, it will receive all further keyboard events, given that the canvas itself also has focus. It's usually best to call focus_set on the canvas whenever you set focus to a canvas item.

To remove focus from the item, call this method with an empty string.

To find out what item that currently has focus, call this method without any arguments.

item
: Item specifier. To remove focus from any item, use an empty string.

Returns:
: If the item specifier is omitted, this method returns the item that currently has focus, or None if no item has focus.

gettags(item) [#]

Gets tags associated with an item.

item
: Item specifier.

Returns:
: A tuple containing all tags associated with the item.

icursor(item, index) [#]

Moves the insertion cursor to the given position. This method can only be used with editable items.

item
: Item specifier.

index
: Cursor index.

index(item, index) [#]

Gets the numerical cursor index corresponding to the given index. Numerical indexes work like Python's sequence indexes; 0 is just to the left of the first character, and len(text) is just to the right of the last character.

item
: Item specifier.

index
: An index. You can use a numerical index, or one of INSERT (the current insertion cursor), END (the length of the text), or SEL_FIRST and SEL_LAST (the selection start and end). You can also use the form "@x,y" where x and y are canvas coordinates, to get the index closest to the given coordinate.

Returns:
: A numerical index (an integer).

insert(item, index, text) [#]

Inserts text into an editable item.

item
: Item specifier.

index
: Where to insert the text. This can be either a numerical index or a symbolic constant. See the description of the **index** method for details. If you insert text at the INSERT index, the cursor is moved along with the text.

text
: The text to insert.

itemcget(item, option) [#]

Gets the current value for an item option.

item
: Item specifier.

option
: Item option.

Returns:
: The option value. If the item specifier refers to more than one item, this method returns the option value for the first item found.

itemconfig(item, **options) [#]

Changes one or more options for all matching items.

item
: Item specifier.

***options*
: Item options.

itemconfigure(item, **options) [#]

Same as **itemconfig**.

lift(item, **options) [#]

(Deprecated) Moves item to top of stack. Same as **tag_raise**.

lower(item, **options) [#]

(Deprecated) Moves item to bottom of stack. Same as **tag_lower**.

move(item, dx, dy) [#]

Moves matching items by an offset.

item

Item specifier.

dx
: Horizontal offset.

dy
: Vertical offset.

postscript(options)** [#]

Generates a Postscript rendering of the canvas contents. Images and embedded widgets are not included.

***options*
: Postscript options.

scale(self, xscale, yscale, xoffset, yoffset) [#]

Resizes matching items by scale factor. The coordinates for each item are recalculated as ((coord-offset)*scale+offset); in other words, each item first moved by -offset, then multiplied with the scale factor, and then moved back again. Note that this method modifies the item coordinates; you may loose precision if you use this method several times on the same items.

xscale
: Horizontal scale.

yscale
: Vertical scale.

xoffset
: Horizontal offset, in canvas coordinates.

yoffset
: Vertical scale, in canvas coordinates.

scan_dragto(x, y) [#]

Scrolls the widget contents relative to the scanning anchor. The contents are moved 10 times the distance between the anchor and the given position. Use **scan_mark** to set the anchor.

x
: The horizontal coordinate.

y
: The vertical coordinate.

scan_mark(x, y) [#]

Sets the scanning anchor. This sets an anchor that can be used for fast scrolling to the given mouse coordinate.

x
: The horizontal coordinate.

y
: The vertical coordinate.

select_adjust(item, index) [#]

Adjusts the selection, so that it includes the given index. This method also sets the selection anchor to this position. This is typically used by mouse bindings.

item
: Item specifier.

index
: Selection index.

select_clear() [#]

Removes the selection, if it is in this canvas widget.

select_from(item, index) [#]

Sets the selection anchor point. Use **select_adjust** or **select_to** to extend the selection.

item
: Item specifier.

index
: Selection anchor.

select_item() [#]

Returns the item that owns the text selection for this canvas widget.

Note that this method always returns None in some older versions of tkinter. To work around this problem, replace the method call with **canvas.tk.call(canvas._w, "select", "item")**.

Returns:
: Item specifier, or None if there's no selection.

select_to(item, index) [#]

Modifies the selection so it includes the region between the current selection anchor and the given index. The anchor is set by **select_from** or **select_adjust**.

item
: Item specifier.

index
: Selection end point.

tag_bind(item, event=None, callback, add=None) [#]

Adds an event binding to all matching items.

Note that the new bindings are associated with the items, not the tag. For example, if you attach bindings to all items having the **movable** tag, they will only be attached to any existing items with that tag. If you create new items tagged as **movable**, they will not get those bindings.

item
: The item specifier (tag or id).

event
: The event specifier.

callback
: A callable object that takes one argument. This callback is called with an event descriptor, for events matching the given event specifier.

add
: Usually, the new binding replaces any existing binding for the same event sequence. If this argument is present, and set to "+", the new binding is added to to any existing binding.

tag_lower(item) [#]

Moves a canvas item to the bottom of the canvas stack. If multiple items match, they are all moved, with their relative order preserved.

This method doesn't work with window items. To change their order, use **lower** on the widget instance instead.

item
: Item specifier.

tag_raise(item) [#]

Moves the given item to the top of the canvas stack. If multiple items match, they are all moved, with their relative order preserved.

This method doesn't work with window items. To change their order, use **lift** on the widget instance instead.

item
: Item specifier.

tag_unbind(self, item, event) [#]

Removes the binding, if any, for the given event sequence. This applies to all matching items.

item
: Item specifier.

sequence
: Event specifier.

tkraise(item, **options) [#]

(Deprecated) Same as **tag_raise**.

type(item) [#]

Returns the type of the given item. If item refers to more than one item, this method returns the type of the first item found.

item
: Item specifier.

Returns:
: A string, giving the item type. This can be one of **"arc"**, **"bitmap"**, **"image"**, **"line"**, **"oval"**, **"polygon"**, **"rectangle"**, **"text"**, or **"window"**.

xview(how, *args) [#]

Adjusts the canvas view horizontally.

how
: How to adjust the canvas. This can be either "moveto" or "scroll".

**args*
: Additional arguments. For the "moveto" method, this is a single fraction. For the "scroll" method, this is a unit and a count. For details, see the descriptions of the **xview_moveto** and **xview_scroll** methods.

xview_moveto(fraction) [#]

Adjusts the canvas so that the given offset is at the left edge of the canvas.

fraction
: Scroll offset. Offset 0.0 is the beginning of the **scrollregion**, 1.0 the end.

xview_scroll(number, what) [#]

Scrolls the canvas horizontally by the given amount.

number
: Number of units.

what
: What unit to use. This can be either **"units"** (small steps) or **"pages"**.

yview(how, *args) [#]

Adjusts the canvas view vertically.

how

How to adjust the canvas. This can be either “moveto” or “scroll”.

**args*
: Additional arguments. For the “moveto” method, this is a single fraction. For the “scroll” method, this is a unit and a count. For details, see the descriptions of the **yview_moveto** and **yview_scroll** methods.

yview_moveto(fraction) [#]

Adjusts the canvas so that the given offset is at the top edge of the canvas.

fraction
: Scroll offset. Offset 0.0 is the beginning of the **scrollregion**, 1.0 the end.

yview_scroll(number, what) [#]

Scrolls the canvas vertically by the given amount.

number
: Number of units.

what
: What unit to use. This can be either **“units”** (small steps) or **“pages”**.