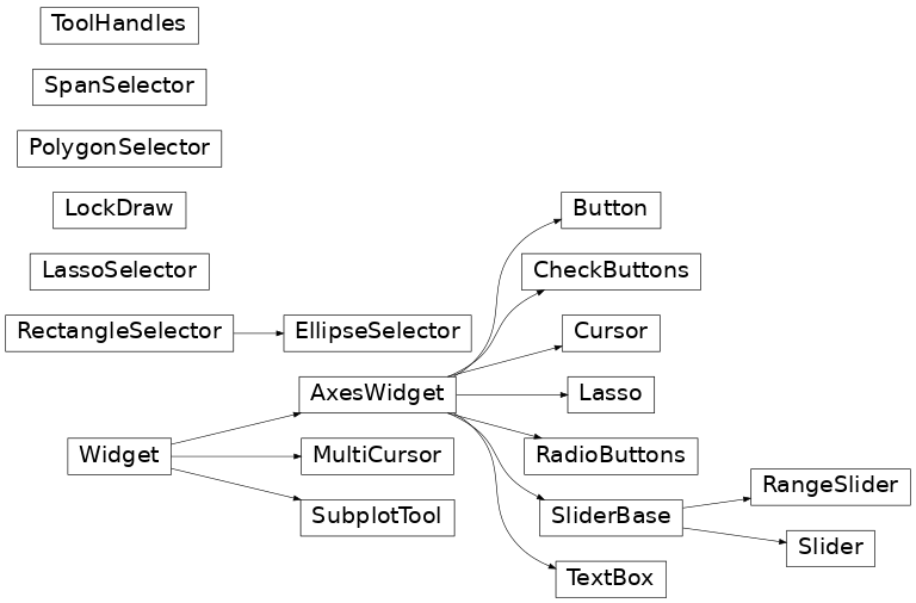
matplotlib.widgets



GUI neutral widgets

Widgets that are designed to work for any of the GUI backends. All of these widgets require you to predefine a `matplotlib.axes.Axes` instance and pass that as the first parameter. Matplotlib doesn't try to be too smart with respect to layout -- you will have to figure out how wide and tall you want your Axes to be to accommodate your widget.

class `matplotlib.widgets.AxesWidget`(*ax*) [source]

Bases: `matplotlib.widgets.Widget`

Widget connected to a single `Axes`.

To guarantee that the widget remains responsive and not garbage-collected, a reference to the object should be maintained by the user.

Table of Contents

matplotlib.widgets

- GUI neutral widgets

Show Page Source

This is necessary because the callback registry maintains only weak-refs to the functions, which are member functions of the widget. If there are no references to the widget object it may be garbage collected which will disconnect the callbacks.

Attributes:

ax : `Axes`

The parent axes for the widget.

canvas : `FigureCanvasBase`

The parent figure canvas for the widget.

`active` : bool

Is the widget active?

---

property `cids`

`connect_event`(*self*, *event*, *callback*) [source]

Connect a callback function with an event.

This should be used in lieu of `figure.canvas.mpl_connect` since this function stores callback ids for later clean up.

`disconnect_events`(*self*) [source]

Disconnect all events created by this widget.

---

class `matplotlib.widgets.Button`(*ax*, *label*, *image=None*, *color='0.85'*, *hovercolor='0.95'*) [source]

Bases: `matplotlib.widgets.AxesWidget`

A GUI neutral button.

For the button to remain responsive you must keep a reference to it. Call `on_clicked` to connect to the button.

Attributes:

ax

The `matplotlib.axes.Axes` the button renders into.

label

A `matplotlib.text.Text` instance.

color

The color of the button when not hovering.

hovercolor

The color of the button when hovering.

Parameters:

ax : `Axes`

The `Axes` instance the button will be placed into.

label : str

The button text.

image : array-like or PIL Image

The image to place in the button, if not *None*. The parameter is directly forwarded to `imshow`.

color : color

The color of the button when not activated.

hovercolor : color

The color of the button when the mouse is over it.

property `cnt`

`disconnect(self, cid)` [source]

Remove the callback function with connection id *cid*.

property `observers`

`on_clicked(self, func)` [source]

Connect the callback function *func* to button click events.

Returns a connection id, which can be used to disconnect the callback.

class `matplotlib.widgets.CheckButtons(ax, labels, actives=None)` [source]

Bases: `matplotlib.widgets.AxesWidget`

A GUI neutral set of check buttons.

For the check buttons to remain responsive you must keep a reference to this object.

Connect to the CheckButtons with the `on_clicked` method.

Attributes:

ax : `Axes`

The parent axes for the widget.

labels : list of `Text`

rectangles : list of `Rectangle`

lines : list of (`Line2D`, `Line2D`) pairs

List of lines for the x's in the check boxes. These lines exist for each box, but have `set_visible(False)` when its box is not checked.

Add check buttons to `matplotlib.axes.Axes` instance *ax*.

Parameters:

ax : `Axes`

The parent axes for the widget.

labels : list of str

The labels of the check buttons.

actives : list of bool, optional

The initial check states of the buttons. The list must have the same length as *labels*. If not given, all buttons are unchecked.

---

property `cnt`

`disconnect(self, cid)` [source]

Remove the observer with connection id *cid*.

`get_status(self)` [source]

Return a tuple of the status (True/False) of all of the check buttons.

---

property `observers`

on_clicked(*self*, *func*) [source]

Connect the callback function *func* to button click events.

Returns a connection id, which can be used to disconnect the callback.

set_active(*self*, *index*) [source]

Toggle (activate or deactivate) a check button by index.

Callbacks will be triggered if `eventson` is True.

Parameters:

index : int

Index of the check button to toggle.

Raises:

ValueError

If *index* is invalid.

class matplotlib.widgets.Cursor(*ax*, *horizOn=True*, *vertOn=True*, *useblit=False*, ***lineprops*) [source]

Bases: matplotlib.widgets.AxesWidget

A crosshair cursor that spans the axes and moves with mouse cursor.

For the cursor to remain responsive you must keep a reference to it.

Parameters:

ax : matplotlib.axes.Axes

The Axes to attach the cursor to.

horizOn : bool, default: True

Whether to draw the horizontal line.

vertOn : bool, default: True

Whether to draw the vertical line.

useblit : bool, default: False

Use blitting for faster drawing if supported by the backend.

Other Parameters: ****lineprops**

`Line2D` properties that control the appearance of the lines. See also `axhline`.

Examples

See Cursor.

`clear`(*self*, *event*) [source]

Internal event handler to clear the cursor.

`onmove`(*self*, *event*) [source]

Internal event handler to draw the cursor when the mouse moves.

---

class `matplotlib.widgets.EllipseSelector`(*ax*, *onselect*, *drawtype='box'*, *minspanx=0*, *minspany=0*, *useblit=False*, *lineprops=None*, *rectprops=None*, *spancoords='data'*, *button=None*, *maxdist=10*, *marker_props=None*, *interactive=False*, *state_modifier_keys=None*) [source]

Bases: `matplotlib.widgets.RectangleSelector`

Select an elliptical region of an axes.

For the cursor to remain responsive you must keep a reference to it.

Example usage:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.widgets import EllipseSelector

def onselect(eclick, erelease):
    "eclick and erelease are matplotlib events at press
    print('startposition: (%f, %f)' % (eclick.xdata, ecl
    print('endposition  : (%f, %f)' % (erelease.xdata, e
    print('used button  : ', eclick.button)

def toggle_selector(event):
    print(' Key pressed.')
    if event.key in ['Q', 'q'] and toggle_selector.ES.ac
        print('EllipseSelector deactivated.')
        toggle_selector.RS.set_active(False)
    if event.key in ['A', 'a'] and not toggle_selector.E
        print('EllipseSelector activated.')
        toggle_selector.ES.set_active(True)

x = np.arange(100.) / 99
y = np.sin(x)
fig, ax = plt.subplots()
ax.plot(x, y)

toggle_selector.ES = EllipseSelector(ax, onselect, drawt
fig.canvas.mpl_connect('key_press_event', toggle_selecto
plt.show()
```

Parameters:

ax : `Axes`

The parent axes for the widget.

onselect : function

A callback function that is called after a selection is completed. It must have the signature:

```
def onselect(eclick: MouseEvent, erelease: MouseEvent)
```

where *eclick* and *erelease* are the mouse click and release `MouseEvent`s that start and complete the selection.

drawtype : {"box", "line", "none"}, default: "box"

Whether to draw the full rectangle box, the diagonal line of the rectangle, or nothing at all.

minspanx : float, default: 0

Selections with an x-span less than *minspanx* are ignored.

minspany : float, default: 0

Selections with an y-span less than *minspany* are ignored.

useblit : bool, default: False

Whether to use blitting for faster drawing (if supported by the backend).

lineprops : dict, optional

Properties with which the line is drawn, if `drawtype == "line"`. Default:

```
dict(color="black", linestyle="-", linewidth=2, alpha=0.5)
```

rectprops : dict, optional

Properties with which the rectangle is drawn, if `drawtype == "box"`. Default:

```
dict(facecolor="red", edgecolor="black", alpha=0.2, fill=True)
```

spancoords : {"data", "pixels"}, default: "data"

Whether to interpret *minspanx* and *minspany* in data or in pixel coordinates.

button : `MouseButton`, list of `MouseButton`, default: all buttons

Button(s) that trigger rectangle selection.

maxdist : float, default: 10

Distance in pixels within which the interactive tool handles can be activated.

marker_props : dict

Properties with which the interactive handles are drawn. Currently not implemented and ignored.

interactive : bool, default: False

Whether to draw a set of handles that allow interaction with the widget after it is drawn.

state_modifier_keys : dict, optional

Keyboard modifiers which affect the widget's behavior. Values amend the defaults.

- "move": Move the existing shape, default: no modifier.
- "clear": Clear the current shape, default: "escape".
- "square": Makes the shape square, default: "shift".
- "center": Make the initial point the center of the shape, default: "ctrl".

"square" and "center" can be combined.

`draw_shape(self, extents)` [source]

---

class `matplotlib.widgets.Lasso(ax, xy, callback=None, useblit=True)` [source]

Bases: `matplotlib.widgets.AxesWidget`

Selection curve of an arbitrary shape.

The selected path can be used in conjunction with `contains_point` to select data points from an image.

Unlike `LassoSelector`, this must be initialized with a starting point *xy*, and the `Lasso` events are destroyed upon release.

Parameters:

ax : `Axes`

The parent axes for the widget.

xy : (float, float)

Coordinates of the start of the lasso.

callback : callable

Whenever the lasso is released, the *callback*

function is called and passed the vertices of the selected path.

onmove(*self*, *event*) [source]

onrelease(*self*, *event*) [source]

class matplotlib.widgets.LassoSelector(*ax*, *onselect=None*, *useblit=True*, *lineprops=None*, *button=None*) [source]

Bases: matplotlib.widgets._SelectorWidget

Selection curve of an arbitrary shape.

For the selector to remain responsive you must keep a reference to it.

The selected path can be used in conjunction with `contains_point` to select data points from an image.

In contrast to `Lasso`, `LassoSelector` is written with an interface similar to `RectangleSelector` and `SpanSelector`, and will continue to interact with the axes until disconnected.

Example usage:

```
ax = plt.subplot()
ax.plot(x, y)

def onselect(verts):
    print(verts)
lasso = LassoSelector(ax, onselect)
```

Parameters:

ax : `Axes`

The parent axes for the widget.

onselect : function

Whenever the lasso is released, the *onselect* function is called and passed

the vertices of the selected path.

button : `MouseButton` or list of `MouseButton`, optional

The mouse buttons used for rectangle selection. Default is `None`, which corresponds to all buttons.

`onpress(self, event)` [source]

`onrelease(self, event)` [source]

class `matplotlib.widgets.LockDraw` [source]

Bases: `object`

Some widgets, like the cursor, draw onto the canvas, and this is not desirable under all circumstances, like when the toolbar is in zoom-to-rect mode and drawing a rectangle. To avoid this, a widget can acquire a canvas' lock with `canvas.widgetlock(widget)` before drawing on the canvas; this will prevent other widgets from doing so at the same time (if they also try to acquire the lock first).

`available(self, o)` [source]

Return whether drawing is available to *o*.

`isowner(self, o)` [source]

Return whether *o* owns this lock.

`locked(self)` [source]

Return whether the lock is currently held by an owner.

`release(self, o)` [source]

Release the lock from *o*.

---

class matplotlib.widgets.MultiCursor(*canvas*, *axes*, *useblit=True*, *horizOn=False*, *vertOn=True*, ***lineprops*) [source]

Bases: matplotlib.widgets.Widget

Provide a vertical (default) and/or horizontal line cursor shared between multiple axes.

For the cursor to remain responsive you must keep a reference to it.

Example usage:

```
from matplotlib.widgets import MultiCursor
import matplotlib.pyplot as plt
import numpy as np

fig, (ax1, ax2) = plt.subplots(nrows=2, sharex=True)
t = np.arange(0.0, 2.0, 0.01)
ax1.plot(t, np.sin(2*np.pi*t))
ax2.plot(t, np.sin(4*np.pi*t))

multi = MultiCursor(fig.canvas, (ax1, ax2), color='r', l
                    horizOn=False, vertOn=True)
plt.show()
```

clear(*self*, *event*) [source]

Clear the cursor.

connect(*self*) [source]

Connect events.

disconnect(*self*) [source]

Disconnect events.

onmove(*self*, *event*) [source]

class matplotlib.widgets.PolygonSelector(*ax*, *onselect*, *useblit=False*, *lineprops=None*, *markerprops=None*, *vertex_select_radius=15*) [source]

Bases: matplotlib.widgets._SelectorWidget

Select a polygon region of an axes.

Place vertices with each mouse click, and make the selection by completing the polygon (clicking on the first vertex). Hold the *ctrl* key and click and drag a vertex to reposition it (the *ctrl* key is not necessary if the polygon has already been completed). Hold the *shift* key and click and drag anywhere in the axes to move all vertices. Press the *esc* key to start a new polygon.

For the selector to remain responsive you must keep a reference to it.

Parameters:

ax : Axes

The parent axes for the widget.

onselect : function

When a polygon is completed or modified after completion, the *onselect* function is called and passed a list of the vertices as `(xdata, ydata)` tuples.

useblit : bool, default: False

lineprops : dict, default: `dict(color='k', linestyle='-', linewidth=2, alpha=0.5)`.

Artist properties for the line representing the edges of the polygon.

markerprops : dict, default: `dict(marker='o', markersize=7, mec='k', mfc='k', alpha=0.5)`.

Artist properties for the markers drawn at the vertices of the polygon.

vertex_select_radius : float, default: 15px

A vertex is selected (to complete the polygon or to move a vertex) if the mouse click is within *vertex_select_radius* pixels of the vertex.

Examples

Polygon Selector

`onmove(self, event)` [source]

Cursor move event handler and validator.

---

property `verts`

The polygon vertices, as a list of `(x, y)` pairs.

---

class `matplotlib.widgets.RadioButtons(ax, labels, active=0, activecolor='blue')` [source]

Bases: `matplotlib.widgets.AxesWidget`

A GUI neutral radio button.

For the buttons to remain responsive you must keep a reference to this object.

Connect to the RadioButtons with the `on_clicked` method.

Attributes:

ax : `Axes`

The parent axes for the widget.

activecolor : color

The color of the selected button.

labels : list of `Text`

The button labels.

circles : list of `Circle`

The buttons.

value_selected : str

The label text of the currently selected button.

Add radio buttons to an `Axes`.

Parameters:

ax : `Axes`

The axes to add the buttons to.

labels : list of str

The button labels.

active : int

The index of the initially selected button.

activecolor : color

The color of the selected button.

---

property `cnt`

`disconnect(self, cid)` [source]

Remove the observer with connection id *cid*.

property `observers`

`on_clicked(self, func)` [source]

Connect the callback function *func* to button click events.

Returns a connection id, which can be used to disconnect the callback.

`set_active(self, index)` [source]

Select button with number *index*.

Callbacks will be triggered if `eventson` is True.

class `matplotlib.widgets.RangeSlider(ax, label, valmin, valmax, valinit=None, valfmt=None, closedmin=True, closedmax=True, dragging=True, valstep=None, orientation='horizontal', **kwargs)` [source]

Bases: `matplotlib.widgets.SliderBase`

A slider representing a range of floating point values. Defines the min and max of the range via the *val* attribute as a tuple of (min, max).

Create a slider that defines a range contained within [*valmin*, *valmax*] in axes *ax*. For the slider to remain responsive you must maintain a reference to it. Call `on_changed()` to connect to the slider event.

Attributes:

val : tuple of float

Slider value.

Parameters:

ax : Axes

The Axes to put the slider in.

label : str

Slider label.

valmin : float

The minimum value of the slider.

valmax : float

The maximum value of the slider.

valinit : tuple of float or None, default: None

The initial positions of the slider. If None the initial positions will be at the 25th and 75th percentiles of the range.

valfmt : str, default: None

%-format string used to format the slider values. If None, a `ScalarFormatter` is used instead.

closedmin : bool, default: True

Whether the slider interval is closed on the bottom.

closedmax : bool, default: True

Whether the slider interval is closed on the top.

dragging : bool, default: True

If True the slider can be dragged by the mouse.

valstep : float, default: None

If given, the slider will snap to multiples of *valstep*.

orientation : {'horizontal', 'vertical'}, default: 'horizontal'

The orientation of the slider.

Notes

Additional kwargs are passed on to `self.poly` which is the `Rectangle` that draws the slider knob. See the `Rectangle` documentation for valid property names (`facecolor`, `edgecolor`, `alpha`, etc.).

`on_changed(self, func)` [source]

Connect *func* as callback function to changes of the slider value.

Parameters: **func** : callable

Function to call when slider is changed. The function must accept a numpy array with shape (2,) as its argument.

Returns: int

Connection id (which can be used to disconnect *func*).

`set_max(self, max)` [source]

Set the lower value of the slider to *max*.

Parameters: **max** : float

`set_min(self, min)` [source]

Set the lower value of the slider to *min*.

Parameters: **min** : float

`set_val(self, val)` [source]

Set slider value to *val*.

Parameters: **val** : tuple or array-like of float

class `matplotlib.widgets.RectangleSelector(ax, onselect, drawtype='box', minspanx=0, minspany=0, useblit=False, lineprops=None, rectprops=None, spancoords='data', button=None, maxdist=10, marker_props=None, interactive=False, state_modifier_keys=None)` [source]

Bases: `matplotlib.widgets._SelectorWidget`

Select a rectangular region of an axes.

For the cursor to remain responsive you must keep a reference to it.

Examples

Rectangle Selector

Parameters: **ax** : `Axes`

The parent axes for the widget.

onselect : function

A callback function that is called after a selection is completed. It must have the signature:

```
def onselect(eclick: MouseEvent, erelease: MouseEvent)
```

where *eclick* and *erelease* are the mouse click and release `MouseEvent`s that start and complete the selection.

drawtype : {"box", "line", "none"}, default: "box"

Whether to draw the full rectangle box, the diagonal line of the rectangle, or nothing at all.

minspanx : float, default: 0

Selections with an x-span less than *minspanx* are ignored.

minspany : float, default: 0

Selections with an y-span less than *minspany* are ignored.

useblit : bool, default: False

Whether to use blitting for faster drawing (if supported by the backend).

lineprops : dict, optional

Properties with which the line is drawn, if `drawtype == "line"`. Default:

```
dict(color="black", linestyle="-", linewidth=2, alpha=0.5)
```

rectprops : dict, optional

Properties with which the rectangle is drawn, if `drawtype == "box"`. Default:

```
dict(facecolor="red", edgecolor="black", alpha=0.2, fill=True)
```

spancoords : {"data", "pixels"}, default: "data"

Whether to interpret *minspanx* and *minspany* in data or in pixel coordinates.

button : `MouseButton`, list of `MouseButton`, default: all buttons

Button(s) that trigger rectangle selection.

maxdist : float, default: 10

Distance in pixels within which the interactive tool handles can be activated.

marker_props : dict

Properties with which the interactive handles are drawn. Currently not implemented and ignored.

interactive : bool, default: False

Whether to draw a set of handles that allow interaction with the widget after it is drawn.

state_modifier_keys : dict, optional

Keyboard modifiers which affect the widget's behavior. Values amend the defaults.

- "move": Move the existing shape, default: no modifier.
- "clear": Clear the current shape, default: "escape".
- "square": Makes the shape square, default: "shift".
- "center": Make the initial point the center of the shape, default: "ctrl".

"square" and "center" can be combined.

---

property `center`

Center of rectangle.

---

property `corners`

Corners of rectangle from lower left, moving clockwise.

`draw_shape`(*`self`*, *`extents`*) [source]

---

property `edge_centers`

Midpoint of rectangle edges from left, moving anti-clockwise.

---

property `extents`

Return (xmin, xmax, ymin, ymax).

---

property `geometry`

Return an array of shape (2, 5) containing the x (`RectangleSelector.geometry[1, :]`) and y (`RectangleSelector.geometry[0, :]`) coordinates of the four corners of the rectangle starting and ending in the top left corner.

---

class `matplotlib.widgets.Slider`(*ax*, *label*, *valmin*, *valmax*, *valinit=0.5*, *valfmt=None*, *closedmin=True*, *closedmax=True*, *slidermin=None*, *slidermax=None*, *dragging=True*, *valstep=None*, *orientation='horizontal'*, ***, *initcolor='r'*, ***kwargs*) [source]

Bases: `matplotlib.widgets.SliderBase`

A slider representing a floating point range.

Create a slider from *valmin* to *valmax* in axes *ax*. For the slider to remain responsive you must maintain a reference to it. Call `on_changed()` to connect to the slider event.

Attributes:

val : float

Slider value.

Parameters:

ax : Axes

The Axes to put the slider in.

label : str

Slider label.

valmin : float

The minimum value of the slider.

valmax : float

The maximum value of the slider.

valinit : float, default: 0.5

The slider initial position.

valfmt : str, default: None

%-format string used to format the slider value. If None, a `ScalarFormatter` is used instead.

closedmin : bool, default: True

Whether the slider interval is closed on the bottom.

closedmax : bool, default: True

Whether the slider interval is closed on the top.

slidermin : Slider, default: None

Do not allow the current slider to have a value less than the value of the Slider *slidermin*.

slidermax : Slider, default: None

Do not allow the current slider to have a value greater than the value of the Slider *slidermax*.

dragging : bool, default: True

If True the slider can be dragged by the mouse.

valstep : float or array-like, default: None

If a float, the slider will snap to multiples of *valstep*. If an array the slider will snap to the values in the array.

orientation : {'horizontal', 'vertical'}, default: 'horizontal'

The orientation of the slider.

initcolor : color, default: 'r'

The color of the line at the *valinit* position. Set to `'none'` for no line.

Notes

Additional kwargs are passed on to `self.poly` which is the `Rectangle` that draws the slider knob. See the `Rectangle` documentation for valid property names (`facecolor`, `edgecolor`, `alpha`, etc.).

---

property `cnt`

---

property `observers`

`on_changed(self, func)` [source]

Connect *func* as callback function to changes of the slider value.

Parameters: **func** : callable

Function to call when slider is changed. The function must accept a single float as its arguments.

Returns: int

Connection id (which can be used to disconnect *func*).

`set_val`(*self*, *val*) [source]

Set slider value to *val*.

Parameters: **val** : float

---

class `matplotlib.widgets.SliderBase`(*ax*, *orientation*, *closedmin*, *closedmax*, *valmin*, *valmax*, *valfmt*, *dragging*, *valstep*) [source]

Bases: `matplotlib.widgets.AxesWidget`

The base class for constructing Slider widgets. Not intended for direct usage.

For the slider to remain responsive you must maintain a reference to it.

`disconnect`(*self*, *cid*) [source]

Remove the observer with connection id *cid*.

Parameters: **cid** : int

Connection id of the observer to be removed.

`reset`(*self*) [source]

Reset the slider to the initial value.

class matplotlib.widgets.SpanSelector(*ax*, *onselect*, *direction*, *minspan=None*, *useblit=False*, *rectprops=None*, *onmove_callback=None*, *span_stays=False*, *button=None*) [source]

Bases: matplotlib.widgets._SelectorWidget

Visually select a min/max range on a single axis and call a function with those values.

To guarantee that the selector remains responsive, keep a reference to it.

In order to turn off the SpanSelector, set `span_selector.active` to False. To turn it back on, set it to True.

Parameters:

ax : matplotlib.axes.Axes

onselect : func(min, max), min/max are floats

direction : {"horizontal", "vertical"}

The direction along which to draw the span selector.

minspan : float, default: None

If selection is less than *minspan*, do not call *onselect*.

useblit : bool, default: False

If True, use the backend-dependent blitting features for faster canvas updates.

rectprops : dict, default: None

Dictionary of matplotlib.patches.Patch properties.

onmove_callback : func(min, max), min/max are floats, default: None

Called on mouse move while the span is being selected.

span_stays : bool, default: False

If True, the span stays visible after the mouse is released.

button : `MouseButton` or list of `MouseButton`

The mouse buttons which activate the span selector.

Examples

```
>>> import matplotlib.pyplot as plt
>>> import matplotlib.widgets as mwidgets
>>> fig, ax = plt.subplots()
>>> ax.plot([1, 2, 3], [10, 50, 100])
>>> def onselect(vmin, vmax):
...     print(vmin, vmax)
>>> rectprops = dict(facecolor='blue', alpha=0.5)
>>> span = mwidgets.SpanSelector(ax, onselect, 'horizont
...                              rectprops=rectprops)
>>> fig.show()
```

See also: Span Selector

`ignore(self, event)` [source]

Return whether *event* should be ignored.

This method should be called at the beginning of any event callback.

`new_axes(self, ax)` [source]

Set SpanSelector to operate on a new Axes.

class matplotlib.widgets.SubplotTool(*targetfig*, *toolfig*) [source]

Bases: matplotlib.widgets.Widget

A tool to adjust the subplot params of a matplotlib.figure.Figure.

Parameters:

targetfig : Figure

The figure instance to adjust.

toolfig : Figure

The figure instance to embed the subplot tool into.

property axbottom

property axhspace

property axleft

property axright

property axtop

property axwspace

funcbottom(*self*, *val*) [source]

[*Deprecated*]

Notes

Deprecated since version 3.3:

funchspace(*self*, *val*) [source]

[*Deprecated*]

Notes

Deprecated since version 3.3:

`funcleft(self, val)` [source]

[*Deprecated*]

Notes

Deprecated since version 3.3:

`funcright(self, val)` [source]

[*Deprecated*]

Notes

Deprecated since version 3.3:

`functop(self, val)` [source]

[*Deprecated*]

Notes

Deprecated since version 3.3:

`funcwspace(self, val)` [source]

[*Deprecated*]

Notes

Deprecated since version 3.3:

class matplotlib.widgets.TextBox(*ax*, *label*, *initial=''*, *color='.95'*, *hovercolor='1'*, *label_pad=0.01*) [source]

Bases: matplotlib.widgets.AxesWidget

A GUI neutral text input box.

For the text box to remain responsive you must keep a reference to it.

Call on_text_change to be updated whenever the text changes.

Call on_submit to be updated whenever the user hits enter or leaves the text entry field.

Attributes:

ax : Axes

The parent axes for the widget.

label : Text

color : color

The color of the text box when not hovering.

hovercolor : color

The color of the text box when hovering.

Parameters:

ax : Axes

The Axes instance the button will be placed into.

label : str

Label for this text box.

initial : str

Initial value in the text box.

color : color

The color of the box.

hovercolor : color

The color of the box when the mouse is over it.

label_pad : float

The distance between the label and the right side of the textbox.

`begin_typing(self, x)` [source]

property `change_observers`

property `cnt`

`disconnect(self, cid)` [source]

Remove the observer with connection id *cid*.

`on_submit(self, func)` [source]

When the user hits enter or leaves the submission box, call this *func* with event.

A connection id is returned which can be used to disconnect.

`on_text_change(self, func)` [source]

When the text changes, call this *func* with event.

A connection id is returned which can be used to disconnect.

property params_to_disable

position_cursor(*self*, *x*) [source]

set_val(*self*, *val*) [source]

stop_typing(*self*) [source]

property submit_observers

property text

class matplotlib.widgets.ToolHandles(*ax*, *x*, *y*, *marker='o'*, *marker_props=None*, *useblit=True*) [source]

Bases: object

Control handles for canvas tools.

Parameters:

ax : matplotlib.axes.Axes

Matplotlib axes where tool handles are displayed.

x, y : 1D arrays

Coordinates of control handles.

marker : str

Shape of marker used to display handle. See matplotlib.pyplot.plot.

marker_props : dict

Additional marker properties. See

matplotlib.lines.Line2D.

closest(*self*, *x*, *y*) [source]

Return index and pixel distance to closest index.

set_animated(*self*, *val*) [source]

set_data(*self*, *pts*, *y=None*) [source]

Set x and y positions of handles.

set_visible(*self*, *val*) [source]

property x

property y

class matplotlib.widgets.Widget [source]

Bases: object

Abstract base class for GUI neutral widgets.

property active

Is the widget active?

drawon = *True*

eventson = *True*

get_active(*self*) [source]

Get whether the widget is active.

`ignore`(*self*, *event*) [source]

Return whether *event* should be ignored.

This method should be called at the beginning of any event callback.

`set_active`(*self*, *active*) [source]

Set whether the widget is active.

Last updated on May 08, 2021. Created using Sphinx 3.2.1. Doc version v3.4.2-2-gf801f04d09-dirty.