bqplot Documentation

Release 0.11.2

Bloomberg LP

Nov 08, 2018

Contents

1 Introduction **1**
1.1 Goals . . . 1
1.2 Installation . . . 1

2 Usage **3**
2.1 Examples . . . 3

3 API Reference Documentation **5**
3.1 BQPlot Package . . . 5

Python Module Index **121**

CHAPTER 1

Introduction

bqplot is a Grammar of Graphics-based interactive plotting framework for the Jupyter notebook.

In bqplot, every single attribute of the plot is an interactive widget. This allows the user to integrate any plot with IPython widgets to create a complex and feature rich GUI from just a few simple lines of Python code.

1.1 Goals

- provide a unified framework for 2-D visualizations with a pythonic API.
- provide a sensible API for adding user interactions (panning, zooming, selection, etc)

Two APIs are provided

- Users can build custom visualizations using the internal object model, which is inspired by the constructs of the Grammar of Graphics (figure, marks, axes, scales), and enrich their visualization with our Interaction Layer.
- Or they can use the context-based API similar to Matplotlib's pyplot, which provides sensible default choices for most parameters.

1.2 Installation

Using pip:

```
pip install bqplot
jupyter nbextension enable --py --sys-prefix bqplot  # can be skipped for notebook␣
↪version 5.3 and above
```

Using conda

```
conda install -c conda-forge bqplot
```

CHAPTER 2

Usage

2.1 Examples

Using the pyplot API

```
import numpy as np
from bqplot import pyplot as plt

plt.figure(1, title='Line Chart')
np.random.seed(0)
n = 200
x = np.linspace(0.0, 10.0, n)
y = np.cumsum(np.random.randn(n))
plt.plot(x, y)
plt.show()
```

[widget] Using the bqplot internal object model

```
import numpy as np
from IPython.display import display
from bqplot import (
    OrdinalScale, LinearScale, Bars, Lines, Axis, Figure
)

size = 20
np.random.seed(0)

x_data = np.arange(size)

x_ord = OrdinalScale()
y_sc = LinearScale()

bar = Bars(x=x_data, y=np.random.randn(2, size), scales={'x': x_ord, 'y':
y_sc}, type='stacked')
```

(continues on next page)

(continued from previous page)

```
line = Lines(x=x_data, y=np.random.randn(size), scales={'x': x_ord, 'y': y_sc},
             stroke_width=3, colors=['red'], display_legend=True, labels=['Line chart
↪'])

ax_x = Axis(scale=x_ord, grid_lines='solid', label='X')
ax_y = Axis(scale=y_sc, orientation='vertical', tick_format='0.2f',
            grid_lines='solid', label='Y')

Figure(marks=[bar, line], axes=[ax_x, ax_y], title='API Example',
       legend_location='bottom-right')
```

[widget]

CHAPTER 3

API Reference Documentation

3.1 BQPlot Package

Each plot starts with a *Figure* object. A *Figure* has a number of *Axis* objects (representing scales) and a number of *Mark* objects. *Mark* objects are a visual representation of the data. Scales transform data into visual properties (typically a number of pixels, a color, etc.).

```
from bqplot import *
from IPython.display import display

x_data = range(10)
y_data = [i ** 2 for i in x_data]

x_sc = LinearScale()
y_sc = LinearScale()

ax_x = Axis(label='Test X', scale=x_sc, tick_format='0.0f')
ax_y = Axis(label='Test Y', scale=y_sc,
            orientation='vertical', tick_format='0.2f')

line = Lines(x=x_data,
             y=y_data,
             scales={'x': x_sc, 'y': y_sc},
             colors=['red', 'yellow'])

fig = Figure(axes=[ax_x, ax_y], marks=[line])

display(fig)
```

3.1.1 Figure

Figure(**kwargs)	Main canvas for drawing a chart.

bqplot.figure.Figure

class bqplot.figure.**Figure**(***kwargs*)

Main canvas for drawing a chart.

The Figure object holds the list of Marks and Axes. It also holds an optional Interaction object that is responsible for figure-level mouse interactions, the "interaction layer".

Besides, the Figure object has two reference scales, for positioning items in an absolute fashion in the figure canvas.

title
title of the figure

Type string (default: '')

axes
list containing the instances of the axes for the figure

Type List of Axes (default: [])

marks
list containing the marks which are to be appended to the figure

Type List of Marks (default: [])

interaction
optional interaction layer for the figure

Type *Interaction* or None (default: None)

scale_x
Scale representing the x values of the figure

Type *Scale*

scale_y
Scale representing the y values of the figure

Type *Scale*

padding_x
Padding to be applied in the horizontal direction of the figure around the data points, proportion of the horizontal length

Type Float (default: 0.0)

padding_y
Padding to be applied in the vertical direction of the figure around the data points, proportion of the vertical length

Type Float (default: 0.025)

legend_location
'bottom-left', 'bottom', 'bottom-right', 'right'} location of the legend relative to the center of the figure

Type {'top-right', 'top', 'top-left', 'left',

background_style
CSS style to be applied to the background of the figure

Type Dict (default: {})

legend_style
CSS style to be applied to the SVG legend e.g, {'fill': 'white'}

Type Dict (default: {})

legend_text
CSS style to be applied to the legend text e.g., {'font-size': 20}

Type Dict (default: {})

title_style
CSS style to be applied to the title of the figure

Type Dict (default: {})

animation_duration
Duration of transition on change of data attributes, in milliseconds.

Type nonnegative int (default: 0)

Layout Attributes

fig_margin
Dictionary containing the top, bottom, left and right margins. The user is responsible for making sure that the width and height are greater than the sum of the margins.

Type dict (default: {top=60, bottom=60, left=60, right=60})

min_aspect_ratio
minimum width / height ratio of the figure

Type float

max_aspect_ratio
maximum width / height ratio of the figure

Type float

save_png:
Saves the figure as a PNG file

save_svg:
Saves the figure as an SVG file

Note: The aspect ratios stand for width / height ratios.

- If the available space is within bounds in terms of min and max aspect ratio, we use the entire available space.
- If the available space is too oblong horizontally, we use the client height and the width that corresponds max_aspect_ratio (maximize width under the constraints).
- If the available space is too oblong vertically, we use the client width and the height that corresponds to min_aspect_ratio (maximize height under the constraint). This corresponds to maximizing the area under the constraints.

Default min and max aspect ratio are both equal to 16 / 9.

__init__(***kwargs*)
Public constructor

Methods

__init__(**kwargs)	Public constructor
add_class(className)	Adds a class to the top level element of the widget.
add_traits(**traits)	Dynamically add trait attributes to the Widget.
class_own_trait_events(name)	Get a dict of all event handlers defined on this class, not a parent.
class_own_traits(**metadata)	Get a dict of all the traitlets defined on this class, not a parent.
class_trait_names(**metadata)	Get a list of all the names of this class' traits.
class_traits(**metadata)	Get a dict of all the traits of this class.
close()	Close method.
close_all()	
get_manager_state([drop_defaults, widgets])	Returns the full state for a widget manager for embedding
get_state([key, drop_defaults])	Gets the widget state, or a piece of it.
get_view_spec()	
handle_comm_opened(comm, msg)	Static method, called when a widget is constructed.
has_trait(name)	Returns True if the object has a trait with the specified name.
hold_sync()	Hold syncing any state until the outermost context manager exits
hold_trait_notifications()	Context manager for bundling trait change notifications and cross validation.
notify_change(change)	Called when a property has changed.
observe(handler[, names, type])	Setup a handler to be called when a trait changes.
on_displayed(callback[, remove])	(Un)Register a widget displayed callback.
on_msg(callback[, remove])	(Un)Register a custom msg receive callback.
on_trait_change([handler, name, remove])	DEPRECATED: Setup a handler to be called when a trait changes.
on_widget_constructed(callback)	Registers a callback to be called when a widget is constructed.
open()	Open a comm to the frontend if one isn't already open.
remove_class(className)	Removes a class from the top level element of the widget.
save_png([filename])	Saves the Figure as a PNG file
save_svg([filename])	Saves the Figure as an SVG file
send(content[, buffers])	Sends a custom msg to the widget model in the front-end.
send_state([key])	Sends the widget state, or a piece of it, to the front-end, if it exists.
set_state(sync_data)	Called when a state is received from the front-end.
set_trait(name, value)	Forcibly sets trait attribute, including read-only attributes.
setup_instance(*args, **kwargs)	This is called **before** self.__init__ is called.
trait_events([name])	Get a dict of all the event handlers of this class.
trait_metadata(traitname, key[, default])	Get metadata values for trait by key.
trait_names(**metadata)	Get a list of all the names of this class' traits.
traits(**metadata)	Get a dict of all the traits of this class.
unobserve(handler[, names, type])	Remove a trait change handler.

Continued on next page

Table 2 – continued from previous page

unobserve_all([name])	Remove trait change handlers of any type for the specified name.

Attributes

animation_duration	An int trait.
axes	An instance of a Python list.
background_style	An instance of a Python dict.
comm	A trait whose value must be an instance of a specified class.
cross_validation_lock	A contextmanager for running a block with our cross validation lock set to True.
fig_margin	An instance of a Python dict.
interaction	A trait whose value must be an instance of a specified class.
keys	An instance of a Python list.
layout	
legend_location	An enum whose value must be in a given sequence.
legend_style	An instance of a Python dict.
legend_text	An instance of a Python dict.
log	A trait whose value must be an instance of a specified class.
marks	An instance of a Python list.
max_aspect_ratio	A float trait.
min_aspect_ratio	A float trait.
model_id	Gets the model id of this widget.
padding_x	A float trait.
padding_y	A float trait.
scale_x	A trait whose value must be an instance of a specified class.
scale_y	A trait whose value must be an instance of a specified class.
title	A trait for unicode strings.
title_style	An instance of a Python dict.
widget_types	
widgets	

3.1.2 Scales

Scale(**kwargs)	The base scale class.
LinearScale(**kwargs)	A linear scale.
LogScale(**kwargs)	A log scale.
DateScale(**kwargs)	A date scale, with customizable formatting.
OrdinalScale(**kwargs)	An ordinal scale.
ColorScale(**kwargs)	A color scale.
DateColorScale(**kwargs)	A date color scale.
OrdinalColorScale(**kwargs)	An ordinal color scale.
GeoScale(**kwargs)	The base projection scale class for Map marks.

Continued on next page

Table 4 – continued from previous page

Mercator(**kwargs)	A geographical projection scale commonly used for world maps.
AlbersUSA(**kwargs)	A composite projection of four Albers projections meant specifically for the United States.
Gnomonic(**kwargs)	A perspective projection which displays great circles as straight lines.
Stereographic(**kwargs)	A perspective projection that uses a bijective and smooth map at every point except the projection point.

bqplot.scales.Scale

class bqplot.scales.**Scale**(***kwargs*)

The base scale class.

Scale objects represent a mapping between data (the domain) and a visual quantity (The range).

scale_types

A registry of existing scale types.

Type dict (class-level attribute)

domain_class

traitlet type used to validate values in of the domain of the scale.

Type type (default: Float)

reverse

whether the scale should be reversed.

Type bool (default: False)

allow_padding

indicates whether figures are allowed to add data padding to this scale or not.

Type bool (default: True)

precedence

attribute used to determine which scale takes precedence in cases when two or more scales have the same rtype and dtype.

Type int (class-level attribute)

__init__(***kwargs*)

Public constructor

Methods

__init__(**kwargs)	Public constructor
add_traits(**traits)	Dynamically add trait attributes to the Widget.
class_own_trait_events(name)	Get a dict of all event handlers defined on this class, not a parent.
class_own_traits(**metadata)	Get a dict of all the traitlets defined on this class, not a parent.
class_trait_names(**metadata)	Get a list of all the names of this class' traits.
class_traits(**metadata)	Get a dict of all the traits of this class.
close()	Close method.

Continued on next page

Table 5 – continued from previous page

close_all()	
get_manager_state([drop_defaults, widgets])	Returns the full state for a widget manager for embedding
get_state([key, drop_defaults])	Gets the widget state, or a piece of it.
get_view_spec()	
handle_comm_opened(comm, msg)	Static method, called when a widget is constructed.
has_trait(name)	Returns True if the object has a trait with the specified name.
hold_sync()	Hold syncing any state until the outermost context manager exits
hold_trait_notifications()	Context manager for bundling trait change notifications and cross validation.
notify_change(change)	Called when a property has changed.
observe(handler[, names, type])	Setup a handler to be called when a trait changes.
on_displayed(callback[, remove])	(Un)Register a widget displayed callback.
on_msg(callback[, remove])	(Un)Register a custom msg receive callback.
on_trait_change([handler, name, remove])	DEPRECATED: Setup a handler to be called when a trait changes.
on_widget_constructed(callback)	Registers a callback to be called when a widget is constructed.
open()	Open a comm to the frontend if one isn't already open.
send(content[, buffers])	Sends a custom msg to the widget model in the front-end.
send_state([key])	Sends the widget state, or a piece of it, to the front-end, if it exists.
set_state(sync_data)	Called when a state is received from the front-end.
set_trait(name, value)	Forcibly sets trait attribute, including read-only attributes.
setup_instance(*args, **kwargs)	This is called **before** self.__init__ is called.
trait_events([name])	Get a dict of all the event handlers of this class.
trait_metadata(traitname, key[, default])	Get metadata values for trait by key.
trait_names(**metadata)	Get a list of all the names of this class' traits.
traits(**metadata)	Get a dict of all the traits of this class.
unobserve(handler[, names, type])	Remove a trait change handler.
unobserve_all([name])	Remove trait change handlers of any type for the specified name.

Attributes

allow_padding	A boolean (True, False) trait.
comm	A trait whose value must be an instance of a specified class.
cross_validation_lock	A contextmanager for running a block with our cross validation lock set to True.
domain_class	A trait whose value must be a subclass of a specified class.
keys	An instance of a Python list.
log	A trait whose value must be an instance of a specified class.

Continued on next page

Table 6 – continued from previous page

model_id	Gets the model id of this widget.
precedence	
reverse	A boolean (True, False) trait.
scale_types	
widget_types	
widgets	

bqplot.scales.LinearScale

class bqplot.scales.**LinearScale**(***kwargs*)

A linear scale.

An affine mapping from a numerical domain to a numerical range.

min

if not None, min is the minimal value of the domain

Type float or None (default: None)

max

if not None, max is the maximal value of the domain

Type float or None (default: None)

rtype

This attribute should not be modifed. The range type of a linear scale is numerical.

Type string (class-level attribute)

dtype

the associated data type / domain type

Type type (class-level attribute)

precedence

attribute used to determine which scale takes precedence in cases when two or more scales have the same rtype and dtype. default_value is 2 because for the same range and domain types, LinearScale should take precedence.

Type int (class-level attribute, default_value=2)

stabilized

if set to False, the domain of the scale is tied to the data range if set to True, the domain of the scale is updated only when the data range is beyond certain thresholds, given by the attributes mid_range and min_range.

Type bool (default: False)

mid_range

Proportion of the range that is spanned initially. Used only if stabilized is True.

Type float (default: 0.8)

min_range

Minimum proportion of the range that should be spanned by the data. If the data span falls beneath that level, the scale is reset. min_range must be <= mid_range. Used only if stabilized is True.

Type float (default: 0.6)

__init__(***kwargs*)

Public constructor

Methods

`__init__`(**kwargs)	Public constructor
`add_traits`(**traits)	Dynamically add trait attributes to the Widget.
`class_own_trait_events`(name)	Get a dict of all event handlers defined on this class, not a parent.
`class_own_traits`(**metadata)	Get a dict of all the traitlets defined on this class, not a parent.
`class_trait_names`(**metadata)	Get a list of all the names of this class' traits.
`class_traits`(**metadata)	Get a `dict` of all the traits of this class.
`close`()	Close method.
`close_all`()	
`get_manager_state`([drop_defaults, widgets])	Returns the full state for a widget manager for embedding
`get_state`([key, drop_defaults])	Gets the widget state, or a piece of it.
`get_view_spec`()	
`handle_comm_opened`(comm, msg)	Static method, called when a widget is constructed.
`has_trait`(name)	Returns True if the object has a trait with the specified name.
`hold_sync`()	Hold syncing any state until the outermost context manager exits
`hold_trait_notifications`()	Context manager for bundling trait change notifications and cross validation.
`notify_change`(change)	Called when a property has changed.
`observe`(handler[, names, type])	Setup a handler to be called when a trait changes.
`on_displayed`(callback[, remove])	(Un)Register a widget displayed callback.
`on_msg`(callback[, remove])	(Un)Register a custom msg receive callback.
`on_trait_change`([handler, name, remove])	DEPRECATED: Setup a handler to be called when a trait changes.
`on_widget_constructed`(callback)	Registers a callback to be called when a widget is constructed.
`open`()	Open a comm to the frontend if one isn't already open.
`send`(content[, buffers])	Sends a custom msg to the widget model in the front-end.
`send_state`([key])	Sends the widget state, or a piece of it, to the front-end, if it exists.
`set_state`(sync_data)	Called when a state is received from the front-end.
`set_trait`(name, value)	Forcibly sets trait attribute, including read-only attributes.
`setup_instance`(*args, **kwargs)	This is called **before** self.__init__ is called.
`trait_events`([name])	Get a `dict` of all the event handlers of this class.
`trait_metadata`(traitname, key[, default])	Get metadata values for trait by key.
`trait_names`(**metadata)	Get a list of all the names of this class' traits.
`traits`(**metadata)	Get a `dict` of all the traits of this class.
`unobserve`(handler[, names, type])	Remove a trait change handler.
`unobserve_all`([name])	Remove trait change handlers of any type for the specified name.

Attributes

allow_padding	A boolean (True, False) trait.
comm	A trait whose value must be an instance of a specified class.
cross_validation_lock	A contextmanager for running a block with our cross validation lock set to True.
domain_class	A trait whose value must be a subclass of a specified class.
keys	An instance of a Python list.
log	A trait whose value must be an instance of a specified class.
max	A float trait.
mid_range	A float trait.
min	A float trait.
min_range	A float trait.
model_id	Gets the model id of this widget.
precedence	
reverse	A boolean (True, False) trait.
rtype	
scale_types	
stabilized	A boolean (True, False) trait.
widget_types	
widgets	

bqplot.scales.LogScale

class bqplot.scales.**LogScale**(***kwargs*)

A log scale.

A logarithmic mapping from a numerical domain to a numerical range.

min

if not None, min is the minimal value of the domain

Type float or None (default: None)

max

if not None, max is the maximal value of the domain

Type float or None (default: None)

rtype

This attribute should not be modifed by the user. The range type of a linear scale is numerical.

Type string (class-level attribute)

dtype

the associated data type / domain type

Type type (class-level attribute)

__init__(***kwargs*)

Public constructor

Methods

`__init__`(**kwargs)	Public constructor
`add_traits`(**traits)	Dynamically add trait attributes to the Widget.
`class_own_trait_events`(name)	Get a dict of all event handlers defined on this class, not a parent.
`class_own_traits`(**metadata)	Get a dict of all the traitlets defined on this class, not a parent.
`class_trait_names`(**metadata)	Get a list of all the names of this class' traits.
`class_traits`(**metadata)	Get a `dict` of all the traits of this class.
`close`()	Close method.
`close_all`()	
`get_manager_state`([drop_defaults, widgets])	Returns the full state for a widget manager for embedding
`get_state`([key, drop_defaults])	Gets the widget state, or a piece of it.
`get_view_spec`()	
`handle_comm_opened`(comm, msg)	Static method, called when a widget is constructed.
`has_trait`(name)	Returns True if the object has a trait with the specified name.
`hold_sync`()	Hold syncing any state until the outermost context manager exits
`hold_trait_notifications`()	Context manager for bundling trait change notifications and cross validation.
`notify_change`(change)	Called when a property has changed.
`observe`(handler[, names, type])	Setup a handler to be called when a trait changes.
`on_displayed`(callback[, remove])	(Un)Register a widget displayed callback.
`on_msg`(callback[, remove])	(Un)Register a custom msg receive callback.
`on_trait_change`([handler, name, remove])	DEPRECATED: Setup a handler to be called when a trait changes.
`on_widget_constructed`(callback)	Registers a callback to be called when a widget is constructed.
`open`()	Open a comm to the frontend if one isn't already open.
`send`(content[, buffers])	Sends a custom msg to the widget model in the front-end.
`send_state`([key])	Sends the widget state, or a piece of it, to the front-end, if it exists.
`set_state`(sync_data)	Called when a state is received from the front-end.
`set_trait`(name, value)	Forcibly sets trait attribute, including read-only attributes.
`setup_instance`(*args, **kwargs)	This is called **before** self.__init__ is called.
`trait_events`([name])	Get a `dict` of all the event handlers of this class.
`trait_metadata`(traitname, key[, default])	Get metadata values for trait by key.
`trait_names`(**metadata)	Get a list of all the names of this class' traits.
`traits`(**metadata)	Get a `dict` of all the traits of this class.
`unobserve`(handler[, names, type])	Remove a trait change handler.
`unobserve_all`([name])	Remove trait change handlers of any type for the specified name.

Attributes

`allow_padding`	A boolean (True, False) trait.

Continued on next page

Table 10 – continued from previous page

comm	A trait whose value must be an instance of a specified class.
cross_validation_lock	A contextmanager for running a block with our cross validation lock set to True.
domain_class	A trait whose value must be a subclass of a specified class.
keys	An instance of a Python list.
log	A trait whose value must be an instance of a specified class.
max	A float trait.
min	A float trait.
model_id	Gets the model id of this widget.
precedence	
reverse	A boolean (True, False) trait.
rtype	
scale_types	
widget_types	
widgets	

bqplot.scales.DateScale

class bqplot.scales.**DateScale**(***kwargs*)

A date scale, with customizable formatting.

An affine mapping from dates to a numerical range.

min

if not None, min is the minimal value of the domain

Type *Date* or None (default: None)

max

if not None, max is the maximal value of the domain

Type *Date* (default: None)

domain_class

traitlet type used to validate values in of the domain of the scale.

Type type (default: Date)

rtype

This attribute should not be modifed by the user. The range type of a linear scale is numerical.

Type string (class-level attribute)

dtype

the associated data type / domain type

Type type (class-level attribute)

__init__(***kwargs*)

Public constructor

Methods

`__init__`(**kwargs)	Public constructor
`add_traits`(**traits)	Dynamically add trait attributes to the Widget.
`class_own_trait_events`(name)	Get a dict of all event handlers defined on this class, not a parent.
`class_own_traits`(**metadata)	Get a dict of all the traitlets defined on this class, not a parent.
`class_trait_names`(**metadata)	Get a list of all the names of this class' traits.
`class_traits`(**metadata)	Get a `dict` of all the traits of this class.
`close`()	Close method.
`close_all`()	
`get_manager_state`([drop_defaults, widgets])	Returns the full state for a widget manager for embedding
`get_state`([key, drop_defaults])	Gets the widget state, or a piece of it.
`get_view_spec`()	
`handle_comm_opened`(comm, msg)	Static method, called when a widget is constructed.
`has_trait`(name)	Returns True if the object has a trait with the specified name.
`hold_sync`()	Hold syncing any state until the outermost context manager exits
`hold_trait_notifications`()	Context manager for bundling trait change notifications and cross validation.
`notify_change`(change)	Called when a property has changed.
`observe`(handler[, names, type])	Setup a handler to be called when a trait changes.
`on_displayed`(callback[, remove])	(Un)Register a widget displayed callback.
`on_msg`(callback[, remove])	(Un)Register a custom msg receive callback.
`on_trait_change`([handler, name, remove])	DEPRECATED: Setup a handler to be called when a trait changes.
`on_widget_constructed`(callback)	Registers a callback to be called when a widget is constructed.
`open`()	Open a comm to the frontend if one isn't already open.
`send`(content[, buffers])	Sends a custom msg to the widget model in the front-end.
`send_state`([key])	Sends the widget state, or a piece of it, to the front-end, if it exists.
`set_state`(sync_data)	Called when a state is received from the front-end.
`set_trait`(name, value)	Forcibly sets trait attribute, including read-only attributes.
`setup_instance`(*args, **kwargs)	This is called **before** self.__init__ is called.
`trait_events`([name])	Get a `dict` of all the event handlers of this class.
`trait_metadata`(traitname, key[, default])	Get metadata values for trait by key.
`trait_names`(**metadata)	Get a list of all the names of this class' traits.
`traits`(**metadata)	Get a `dict` of all the traits of this class.
`unobserve`(handler[, names, type])	Remove a trait change handler.
`unobserve_all`([name])	Remove trait change handlers of any type for the specified name.

Attributes

`allow_padding`	A boolean (True, False) trait.

Continued on next page

Table 12 – continued from previous page

comm	A trait whose value must be an instance of a specified class.
cross_validation_lock	A contextmanager for running a block with our cross validation lock set to True.
domain_class	A trait whose value must be a subclass of a specified class.
keys	An instance of a Python list.
log	A trait whose value must be an instance of a specified class.
max	A datetime trait type.
min	A datetime trait type.
model_id	Gets the model id of this widget.
precedence	
reverse	A boolean (True, False) trait.
rtype	
scale_types	
widget_types	
widgets	

bqplot.scales.OrdinalScale

class bqplot.scales.**OrdinalScale**(***kwargs*)

An ordinal scale.

A mapping from a discrete set of values to a numerical range.

domain

The discrete values mapped by the ordinal scale

Type list (default: [])

rtype

This attribute should not be modifed by the user. The range type of a linear scale is numerical.

Type string (class-level attribute)

dtype

the associated data type / domain type

Type type (class-level attribute)

__init__(***kwargs*)

Public constructor

Methods

__init__(**kwargs)	Public constructor
add_traits(**traits)	Dynamically add trait attributes to the Widget.
class_own_trait_events(name)	Get a dict of all event handlers defined on this class, not a parent.
class_own_traits(**metadata)	Get a dict of all the traitlets defined on this class, not a parent.
class_trait_names(**metadata)	Get a list of all the names of this class' traits.
class_traits(**metadata)	Get a dict of all the traits of this class.

Continued on next page

Table 13 – continued from previous page

close()	Close method.
close_all()	
get_manager_state([drop_defaults, widgets])	Returns the full state for a widget manager for embedding
get_state([key, drop_defaults])	Gets the widget state, or a piece of it.
get_view_spec()	
handle_comm_opened(comm, msg)	Static method, called when a widget is constructed.
has_trait(name)	Returns True if the object has a trait with the specified name.
hold_sync()	Hold syncing any state until the outermost context manager exits
hold_trait_notifications()	Context manager for bundling trait change notifications and cross validation.
notify_change(change)	Called when a property has changed.
observe(handler[, names, type])	Setup a handler to be called when a trait changes.
on_displayed(callback[, remove])	(Un)Register a widget displayed callback.
on_msg(callback[, remove])	(Un)Register a custom msg receive callback.
on_trait_change([handler, name, remove])	DEPRECATED: Setup a handler to be called when a trait changes.
on_widget_constructed(callback)	Registers a callback to be called when a widget is constructed.
open()	Open a comm to the frontend if one isn't already open.
send(content[, buffers])	Sends a custom msg to the widget model in the front-end.
send_state([key])	Sends the widget state, or a piece of it, to the front-end, if it exists.
set_state(sync_data)	Called when a state is received from the front-end.
set_trait(name, value)	Forcibly sets trait attribute, including read-only attributes.
setup_instance(*args, **kwargs)	This is called **before** self.__init__ is called.
trait_events([name])	Get a dict of all the event handlers of this class.
trait_metadata(traitname, key[, default])	Get metadata values for trait by key.
trait_names(**metadata)	Get a list of all the names of this class' traits.
traits(**metadata)	Get a dict of all the traits of this class.
unobserve(handler[, names, type])	Remove a trait change handler.
unobserve_all([name])	Remove trait change handlers of any type for the specified name.

Attributes

allow_padding	A boolean (True, False) trait.
comm	A trait whose value must be an instance of a specified class.
cross_validation_lock	A contextmanager for running a block with our cross validation lock set to True.
domain	An instance of a Python list.
domain_class	A trait whose value must be a subclass of a specified class.
keys	An instance of a Python list.

Continued on next page

Table 14 – continued from previous page

log	A trait whose value must be an instance of a specified class.
model_id	Gets the model id of this widget.
precedence	
reverse	A boolean (True, False) trait.
rtype	
scale_types	
widget_types	
widgets	

bqplot.scales.ColorScale

class bqplot.scales.**ColorScale**(***kwargs*)

A color scale.

A mapping from numbers to colors. The relation is affine by part.

scale_type

scale type

Type {'linear'}

colors

list of colors

Type list of colors (default: [])

min

if not None, min is the minimal value of the domain

Type float or None (default: None)

max

if not None, max is the maximal value of the domain

Type float or None (default: None)

mid

if not None, mid is the value corresponding to the mid color.

Type float or None (default: None)

scheme

Colorbrewer color scheme of the color scale.

Type string (default: 'RdYlGn')

rtype

The range type of a color scale is 'Color'. This should not be modifed.

Type string (class-level attribute)

dtype

the associated data type / domain type

Type type (class-level attribute)

__init__(***kwargs*)

Public constructor

Methods

`__init__`(**kwargs)	Public constructor
`add_traits`(**traits)	Dynamically add trait attributes to the Widget.
`class_own_trait_events`(name)	Get a dict of all event handlers defined on this class, not a parent.
`class_own_traits`(**metadata)	Get a dict of all the traitlets defined on this class, not a parent.
`class_trait_names`(**metadata)	Get a list of all the names of this class' traits.
`class_traits`(**metadata)	Get a `dict` of all the traits of this class.
`close`()	Close method.
`close_all`()	
`get_manager_state`([drop_defaults, widgets])	Returns the full state for a widget manager for embedding
`get_state`([key, drop_defaults])	Gets the widget state, or a piece of it.
`get_view_spec`()	
`handle_comm_opened`(comm, msg)	Static method, called when a widget is constructed.
`has_trait`(name)	Returns True if the object has a trait with the specified name.
`hold_sync`()	Hold syncing any state until the outermost context manager exits
`hold_trait_notifications`()	Context manager for bundling trait change notifications and cross validation.
`notify_change`(change)	Called when a property has changed.
`observe`(handler[, names, type])	Setup a handler to be called when a trait changes.
`on_displayed`(callback[, remove])	(Un)Register a widget displayed callback.
`on_msg`(callback[, remove])	(Un)Register a custom msg receive callback.
`on_trait_change`([handler, name, remove])	DEPRECATED: Setup a handler to be called when a trait changes.
`on_widget_constructed`(callback)	Registers a callback to be called when a widget is constructed.
`open`()	Open a comm to the frontend if one isn't already open.
`send`(content[, buffers])	Sends a custom msg to the widget model in the front-end.
`send_state`([key])	Sends the widget state, or a piece of it, to the front-end, if it exists.
`set_state`(sync_data)	Called when a state is received from the front-end.
`set_trait`(name, value)	Forcibly sets trait attribute, including read-only attributes.
`setup_instance`(*args, **kwargs)	This is called **before** self.__init__ is called.
`trait_events`([name])	Get a `dict` of all the event handlers of this class.
`trait_metadata`(traitname, key[, default])	Get metadata values for trait by key.
`trait_names`(**metadata)	Get a list of all the names of this class' traits.
`traits`(**metadata)	Get a `dict` of all the traits of this class.
`unobserve`(handler[, names, type])	Remove a trait change handler.
`unobserve_all`([name])	Remove trait change handlers of any type for the specified name.

Attributes

allow_padding	A boolean (True, False) trait.
colors	An instance of a Python list.
comm	A trait whose value must be an instance of a specified class.
cross_validation_lock	A contextmanager for running a block with our cross validation lock set to True.
domain_class	A trait whose value must be a subclass of a specified class.
keys	An instance of a Python list.
log	A trait whose value must be an instance of a specified class.
max	A float trait.
mid	A float trait.
min	A float trait.
model_id	Gets the model id of this widget.
precedence	
reverse	A boolean (True, False) trait.
rtype	
scale_type	An enum whose value must be in a given sequence.
scale_types	
scheme	A trait for unicode strings.
widget_types	
widgets	

bqplot.scales.DateColorScale

class bqplot.scales.**DateColorScale**(***kwargs*)

A date color scale.

A mapping from dates to a numerical domain.

min

if not None, min is the minimal value of the domain

Type *Date* or None (default: None)

max

if not None, max is the maximal value of the domain

Type *Date* or None (default: None)

mid

if not None, mid is the value corresponding to the mid color.

Type *Date* or None (default: None)

rtype

This attribute should not be modifed by the user. The range type of a color scale is 'Color'.

Type string (class-level attribute)

dtype

the associated data type / domain type

Type type (class-level attribute)

__init__(***kwargs*)

Public constructor

Methods

`__init__`(**kwargs)	Public constructor
`add_traits`(**traits)	Dynamically add trait attributes to the Widget.
`class_own_trait_events`(name)	Get a dict of all event handlers defined on this class, not a parent.
`class_own_traits`(**metadata)	Get a dict of all the traitlets defined on this class, not a parent.
`class_trait_names`(**metadata)	Get a list of all the names of this class' traits.
`class_traits`(**metadata)	Get a `dict` of all the traits of this class.
`close`()	Close method.
`close_all`()	
`get_manager_state`([drop_defaults, widgets])	Returns the full state for a widget manager for embedding
`get_state`([key, drop_defaults])	Gets the widget state, or a piece of it.
`get_view_spec`()	
`handle_comm_opened`(comm, msg)	Static method, called when a widget is constructed.
`has_trait`(name)	Returns True if the object has a trait with the specified name.
`hold_sync`()	Hold syncing any state until the outermost context manager exits
`hold_trait_notifications`()	Context manager for bundling trait change notifications and cross validation.
`notify_change`(change)	Called when a property has changed.
`observe`(handler[, names, type])	Setup a handler to be called when a trait changes.
`on_displayed`(callback[, remove])	(Un)Register a widget displayed callback.
`on_msg`(callback[, remove])	(Un)Register a custom msg receive callback.
`on_trait_change`([handler, name, remove])	DEPRECATED: Setup a handler to be called when a trait changes.
`on_widget_constructed`(callback)	Registers a callback to be called when a widget is constructed.
`open`()	Open a comm to the frontend if one isn't already open.
`send`(content[, buffers])	Sends a custom msg to the widget model in the front-end.
`send_state`([key])	Sends the widget state, or a piece of it, to the front-end, if it exists.
`set_state`(sync_data)	Called when a state is received from the front-end.
`set_trait`(name, value)	Forcibly sets trait attribute, including read-only attributes.
`setup_instance`(*args, **kwargs)	This is called **before** self.__init__ is called.
`trait_events`([name])	Get a `dict` of all the event handlers of this class.
`trait_metadata`(traitname, key[, default])	Get metadata values for trait by key.
`trait_names`(**metadata)	Get a list of all the names of this class' traits.
`traits`(**metadata)	Get a `dict` of all the traits of this class.
`unobserve`(handler[, names, type])	Remove a trait change handler.
`unobserve_all`([name])	Remove trait change handlers of any type for the specified name.

Attributes

allow_padding	A boolean (True, False) trait.
colors	An instance of a Python list.
comm	A trait whose value must be an instance of a specified class.
cross_validation_lock	A contextmanager for running a block with our cross validation lock set to True.
domain_class	A trait whose value must be a subclass of a specified class.
keys	An instance of a Python list.
log	A trait whose value must be an instance of a specified class.
max	A datetime trait type.
mid	A datetime trait type.
min	A datetime trait type.
model_id	Gets the model id of this widget.
precedence	
reverse	A boolean (True, False) trait.
rtype	
scale_type	An enum whose value must be in a given sequence.
scale_types	
scheme	A trait for unicode strings.
widget_types	
widgets	

bqplot.scales.OrdinalColorScale

class bqplot.scales.**OrdinalColorScale**(***kwargs*)

An ordinal color scale.

A mapping from a discrete set of values to colors.

domain

The discrete values mapped by the ordinal scales.

Type list (default: [])

rtype

This attribute should not be modifed by the user. The range type of a color scale is 'color'.

Type string (class-level attribute)

dtype

the associated data type / domain type

Type type (class-level attribute)

__init__(***kwargs*)

Public constructor

Methods

__init__(**kwargs)	Public constructor
add_traits(**traits)	Dynamically add trait attributes to the Widget.

Continued on next page

Table 19 – continued from previous page

class_own_trait_events(name)	Get a dict of all event handlers defined on this class, not a parent.
class_own_traits(**metadata)	Get a dict of all the traitlets defined on this class, not a parent.
class_trait_names(**metadata)	Get a list of all the names of this class' traits.
class_traits(**metadata)	Get a dict of all the traits of this class.
close()	Close method.
close_all()	
get_manager_state([drop_defaults, widgets])	Returns the full state for a widget manager for embedding
get_state([key, drop_defaults])	Gets the widget state, or a piece of it.
get_view_spec()	
handle_comm_opened(comm, msg)	Static method, called when a widget is constructed.
has_trait(name)	Returns True if the object has a trait with the specified name.
hold_sync()	Hold syncing any state until the outermost context manager exits
hold_trait_notifications()	Context manager for bundling trait change notifications and cross validation.
notify_change(change)	Called when a property has changed.
observe(handler[, names, type])	Setup a handler to be called when a trait changes.
on_displayed(callback[, remove])	(Un)Register a widget displayed callback.
on_msg(callback[, remove])	(Un)Register a custom msg receive callback.
on_trait_change([handler, name, remove])	DEPRECATED: Setup a handler to be called when a trait changes.
on_widget_constructed(callback)	Registers a callback to be called when a widget is constructed.
open()	Open a comm to the frontend if one isn't already open.
send(content[, buffers])	Sends a custom msg to the widget model in the front-end.
send_state([key])	Sends the widget state, or a piece of it, to the front-end, if it exists.
set_state(sync_data)	Called when a state is received from the front-end.
set_trait(name, value)	Forcibly sets trait attribute, including read-only attributes.
setup_instance(*args, **kwargs)	This is called **before** self.__init__ is called.
trait_events([name])	Get a dict of all the event handlers of this class.
trait_metadata(traitname, key[, default])	Get metadata values for trait by key.
trait_names(**metadata)	Get a list of all the names of this class' traits.
traits(**metadata)	Get a dict of all the traits of this class.
unobserve(handler[, names, type])	Remove a trait change handler.
unobserve_all([name])	Remove trait change handlers of any type for the specified name.

Attributes

allow_padding	A boolean (True, False) trait.
colors	An instance of a Python list.

Continued on next page

Table 20 – continued from previous page

comm	A trait whose value must be an instance of a specified class.
cross_validation_lock	A contextmanager for running a block with our cross validation lock set to True.
domain	An instance of a Python list.
domain_class	A trait whose value must be a subclass of a specified class.
keys	An instance of a Python list.
log	A trait whose value must be an instance of a specified class.
max	A float trait.
mid	A float trait.
min	A float trait.
model_id	Gets the model id of this widget.
precedence	
reverse	A boolean (True, False) trait.
rtype	
scale_type	An enum whose value must be in a given sequence.
scale_types	
scheme	A trait for unicode strings.
widget_types	
widgets	

bqplot.scales.GeoScale

class bqplot.scales.**GeoScale**(***kwargs*)

The base projection scale class for Map marks.

The GeoScale represents a mapping between topographic data and a 2d visual representation.

__init__(***kwargs*)

Public constructor

Methods

__init__(**kwargs)	Public constructor
add_traits(**traits)	Dynamically add trait attributes to the Widget.
class_own_trait_events(name)	Get a dict of all event handlers defined on this class, not a parent.
class_own_traits(**metadata)	Get a dict of all the traitlets defined on this class, not a parent.
class_trait_names(**metadata)	Get a list of all the names of this class' traits.
class_traits(**metadata)	Get a dict of all the traits of this class.
close()	Close method.
close_all()	
get_manager_state([drop_defaults, widgets])	Returns the full state for a widget manager for embedding
get_state([key, drop_defaults])	Gets the widget state, or a piece of it.
get_view_spec()	
handle_comm_opened(comm, msg)	Static method, called when a widget is constructed.

Continued on next page

Table 21 – continued from previous page

has_trait(name)	Returns True if the object has a trait with the specified name.
hold_sync()	Hold syncing any state until the outermost context manager exits
hold_trait_notifications()	Context manager for bundling trait change notifications and cross validation.
notify_change(change)	Called when a property has changed.
observe(handler[, names, type])	Setup a handler to be called when a trait changes.
on_displayed(callback[, remove])	(Un)Register a widget displayed callback.
on_msg(callback[, remove])	(Un)Register a custom msg receive callback.
on_trait_change([handler, name, remove])	DEPRECATED: Setup a handler to be called when a trait changes.
on_widget_constructed(callback)	Registers a callback to be called when a widget is constructed.
open()	Open a comm to the frontend if one isn't already open.
send(content[, buffers])	Sends a custom msg to the widget model in the front-end.
send_state([key])	Sends the widget state, or a piece of it, to the front-end, if it exists.
set_state(sync_data)	Called when a state is received from the front-end.
set_trait(name, value)	Forcibly sets trait attribute, including read-only attributes.
setup_instance(*args, **kwargs)	This is called **before** self.__init__ is called.
trait_events([name])	Get a dict of all the event handlers of this class.
trait_metadata(traitname, key[, default])	Get metadata values for trait by key.
trait_names(**metadata)	Get a list of all the names of this class' traits.
traits(**metadata)	Get a dict of all the traits of this class.
unobserve(handler[, names, type])	Remove a trait change handler.
unobserve_all([name])	Remove trait change handlers of any type for the specified name.

Attributes

allow_padding	A boolean (True, False) trait.
comm	A trait whose value must be an instance of a specified class.
cross_validation_lock	A contextmanager for running a block with our cross validation lock set to True.
domain_class	A trait whose value must be a subclass of a specified class.
keys	An instance of a Python list.
log	A trait whose value must be an instance of a specified class.
model_id	Gets the model id of this widget.
precedence	
reverse	A boolean (True, False) trait.
scale_types	
widget_types	
widgets	

bqplot.scales.Mercator

class bqplot.scales.**Mercator**(***kwargs*)

A geographical projection scale commonly used for world maps.

The Mercator projection is a cylindrical map projection which ensures that any course of constant bearing is a straight line.

scale_factor

Specifies the scale value for the projection

Type float (default: 190)

center

Specifies the longitude and latitude where the map is centered.

Type tuple (default: (0, 60))

rotate

Degree of rotation in each axis.

Type tuple (default: (0, 0))

rtype

This attribute should not be modifed. The range type of a geo scale is a tuple.

Type (Number, Number) (class-level attribute)

dtype

the associated data type / domain type

Type type (class-level attribute)

__init__(***kwargs*)

Public constructor

Methods

__init__(**kwargs)	Public constructor
add_traits(**traits)	Dynamically add trait attributes to the Widget.
class_own_trait_events(name)	Get a dict of all event handlers defined on this class, not a parent.
class_own_traits(**metadata)	Get a dict of all the traitlets defined on this class, not a parent.
class_trait_names(**metadata)	Get a list of all the names of this class' traits.
class_traits(**metadata)	Get a dict of all the traits of this class.
close()	Close method.
close_all()	
get_manager_state([drop_defaults, widgets])	Returns the full state for a widget manager for embedding
get_state([key, drop_defaults])	Gets the widget state, or a piece of it.
get_view_spec()	
handle_comm_opened(comm, msg)	Static method, called when a widget is constructed.
has_trait(name)	Returns True if the object has a trait with the specified name.
hold_sync()	Hold syncing any state until the outermost context manager exits

Continued on next page

Table 23 – continued from previous page

hold_trait_notifications()	Context manager for bundling trait change notifications and cross validation.
notify_change(change)	Called when a property has changed.
observe(handler[, names, type])	Setup a handler to be called when a trait changes.
on_displayed(callback[, remove])	(Un)Register a widget displayed callback.
on_msg(callback[, remove])	(Un)Register a custom msg receive callback.
on_trait_change([handler, name, remove])	DEPRECATED: Setup a handler to be called when a trait changes.
on_widget_constructed(callback)	Registers a callback to be called when a widget is constructed.
open()	Open a comm to the frontend if one isn't already open.
send(content[, buffers])	Sends a custom msg to the widget model in the front-end.
send_state([key])	Sends the widget state, or a piece of it, to the front-end, if it exists.
set_state(sync_data)	Called when a state is received from the front-end.
set_trait(name, value)	Forcibly sets trait attribute, including read-only attributes.
setup_instance(*args, **kwargs)	This is called **before** self.__init__ is called.
trait_events([name])	Get a dict of all the event handlers of this class.
trait_metadata(traitname, key[, default])	Get metadata values for trait by key.
trait_names(**metadata)	Get a list of all the names of this class' traits.
traits(**metadata)	Get a dict of all the traits of this class.
unobserve(handler[, names, type])	Remove a trait change handler.
unobserve_all([name])	Remove trait change handlers of any type for the specified name.

Attributes

allow_padding	A boolean (True, False) trait.
center	An instance of a Python tuple.
comm	A trait whose value must be an instance of a specified class.
cross_validation_lock	A contextmanager for running a block with our cross validation lock set to True.
domain_class	A trait whose value must be a subclass of a specified class.
keys	An instance of a Python list.
log	A trait whose value must be an instance of a specified class.
model_id	Gets the model id of this widget.
precedence	
reverse	A boolean (True, False) trait.
rotate	An instance of a Python tuple.
rtype	
scale_factor	A float trait.
scale_types	
widget_types	
widgets	

bqplot.scales.AlbersUSA

class bqplot.scales.**AlbersUSA**(***kwargs*)

A composite projection of four Albers projections meant specifically for the United States.

scale_factor

Specifies the scale value for the projection

Type float (default: 1200)

translate

Type tuple (default: (600, 490))

rtype

This attribute should not be modifed. The range type of a geo scale is a tuple.

Type (Number, Number) (class-level attribute)

dtype

the associated data type / domain type

Type type (class-level attribute)

__init__(***kwargs*)

Public constructor

Methods

__init__(**kwargs)	Public constructor
add_traits(**traits)	Dynamically add trait attributes to the Widget.
class_own_trait_events(name)	Get a dict of all event handlers defined on this class, not a parent.
class_own_traits(**metadata)	Get a dict of all the traitlets defined on this class, not a parent.
class_trait_names(**metadata)	Get a list of all the names of this class' traits.
class_traits(**metadata)	Get a dict of all the traits of this class.
close()	Close method.
close_all()	
get_manager_state([drop_defaults, widgets])	Returns the full state for a widget manager for embedding
get_state([key, drop_defaults])	Gets the widget state, or a piece of it.
get_view_spec()	
handle_comm_opened(comm, msg)	Static method, called when a widget is constructed.
has_trait(name)	Returns True if the object has a trait with the specified name.
hold_sync()	Hold syncing any state until the outermost context manager exits
hold_trait_notifications()	Context manager for bundling trait change notifications and cross validation.
notify_change(change)	Called when a property has changed.
observe(handler[, names, type])	Setup a handler to be called when a trait changes.
on_displayed(callback[, remove])	(Un)Register a widget displayed callback.
on_msg(callback[, remove])	(Un)Register a custom msg receive callback.

Continued on next page

Table 25 – continued from previous page

on_trait_change([handler, name, remove])	DEPRECATED: Setup a handler to be called when a trait changes.
on_widget_constructed(callback)	Registers a callback to be called when a widget is constructed.
open()	Open a comm to the frontend if one isn't already open.
send(content[, buffers])	Sends a custom msg to the widget model in the front-end.
send_state([key])	Sends the widget state, or a piece of it, to the front-end, if it exists.
set_state(sync_data)	Called when a state is received from the front-end.
set_trait(name, value)	Forcibly sets trait attribute, including read-only attributes.
setup_instance(*args, **kwargs)	This is called **before** self.__init__ is called.
trait_events([name])	Get a dict of all the event handlers of this class.
trait_metadata(traitname, key[, default])	Get metadata values for trait by key.
trait_names(**metadata)	Get a list of all the names of this class' traits.
traits(**metadata)	Get a dict of all the traits of this class.
unobserve(handler[, names, type])	Remove a trait change handler.
unobserve_all([name])	Remove trait change handlers of any type for the specified name.

Attributes

allow_padding	A boolean (True, False) trait.
comm	A trait whose value must be an instance of a specified class.
cross_validation_lock	A contextmanager for running a block with our cross validation lock set to True.
domain_class	A trait whose value must be a subclass of a specified class.
keys	An instance of a Python list.
log	A trait whose value must be an instance of a specified class.
model_id	Gets the model id of this widget.
precedence	
reverse	A boolean (True, False) trait.
rtype	
scale_factor	A float trait.
scale_types	
translate	An instance of a Python tuple.
widget_types	
widgets	

bqplot.scales.Gnomonic

class bqplot.scales.**Gnomonic**(***kwargs*)

A perspective projection which displays great circles as straight lines.

The projection is neither equal-area nor conformal.

scale_factor

Specifies the scale value for the projection

Type float (default: 145)

center
Specifies the longitude and latitude where the map is centered.

Type tuple (default: (0, 60))

precision
Specifies the threshold for the projections adaptive resampling to the specified value in pixels.

Type float (default: 0.1)

clip_angle
Specifies the clipping circle radius to the specified angle in degrees.

Type float (default: 89.999)

__init__(***kwargs*)
Public constructor

Methods

__init__(**kwargs)	Public constructor
add_traits(**traits)	Dynamically add trait attributes to the Widget.
class_own_trait_events(name)	Get a dict of all event handlers defined on this class, not a parent.
class_own_traits(**metadata)	Get a dict of all the traitlets defined on this class, not a parent.
class_trait_names(**metadata)	Get a list of all the names of this class' traits.
class_traits(**metadata)	Get a dict of all the traits of this class.
close()	Close method.
close_all()	
get_manager_state([drop_defaults, widgets])	Returns the full state for a widget manager for embedding
get_state([key, drop_defaults])	Gets the widget state, or a piece of it.
get_view_spec()	
handle_comm_opened(comm, msg)	Static method, called when a widget is constructed.
has_trait(name)	Returns True if the object has a trait with the specified name.
hold_sync()	Hold syncing any state until the outermost context manager exits
hold_trait_notifications()	Context manager for bundling trait change notifications and cross validation.
notify_change(change)	Called when a property has changed.
observe(handler[, names, type])	Setup a handler to be called when a trait changes.
on_displayed(callback[, remove])	(Un)Register a widget displayed callback.
on_msg(callback[, remove])	(Un)Register a custom msg receive callback.
on_trait_change([handler, name, remove])	DEPRECATED: Setup a handler to be called when a trait changes.
on_widget_constructed(callback)	Registers a callback to be called when a widget is constructed.
open()	Open a comm to the frontend if one isn't already open.

Continued on next page

Table 27 – continued from previous page

send(content[, buffers])	Sends a custom msg to the widget model in the front-end.
send_state([key])	Sends the widget state, or a piece of it, to the front-end, if it exists.
set_state(sync_data)	Called when a state is received from the front-end.
set_trait(name, value)	Forcibly sets trait attribute, including read-only attributes.
setup_instance(*args, **kwargs)	This is called **before** self.__init__ is called.
trait_events([name])	Get a dict of all the event handlers of this class.
trait_metadata(traitname, key[, default])	Get metadata values for trait by key.
trait_names(**metadata)	Get a list of all the names of this class' traits.
traits(**metadata)	Get a dict of all the traits of this class.
unobserve(handler[, names, type])	Remove a trait change handler.
unobserve_all([name])	Remove trait change handlers of any type for the specified name.

Attributes

allow_padding	A boolean (True, False) trait.
center	An instance of a Python tuple.
clip_angle	A float trait.
comm	A trait whose value must be an instance of a specified class.
cross_validation_lock	A contextmanager for running a block with our cross validation lock set to True.
domain_class	A trait whose value must be a subclass of a specified class.
keys	An instance of a Python list.
log	A trait whose value must be an instance of a specified class.
model_id	Gets the model id of this widget.
precedence	
precision	A float trait.
reverse	A boolean (True, False) trait.
rtype	
scale_factor	A float trait.
scale_types	
widget_types	
widgets	

bqplot.scales.Stereographic

class bqplot.scales.**Stereographic**(***kwargs*)

A perspective projection that uses a bijective and smooth map at every point except the projection point.

The projection is not an equal-area projection but it is conformal.

scale_factor

Specifies the scale value for the projection

Type float (default: 250)

rotate
Degree of rotation in each axis.

Type tuple (default: (96, 0))

center
Specifies the longitude and latitude where the map is centered.

Type tuple (default: (0, 60))

precision
Specifies the threshold for the projections adaptive resampling to the specified value in pixels.

Type float (default: 0.1)

clip_angle
Specifies the clipping circle radius to the specified angle in degrees.

Type float (default: 90.)

__init__ (***kwargs*)
Public constructor

Methods

__init__(**kwargs)	Public constructor
add_traits(**traits)	Dynamically add trait attributes to the Widget.
class_own_trait_events(name)	Get a dict of all event handlers defined on this class, not a parent.
class_own_traits(**metadata)	Get a dict of all the traitlets defined on this class, not a parent.
class_trait_names(**metadata)	Get a list of all the names of this class' traits.
class_traits(**metadata)	Get a dict of all the traits of this class.
close()	Close method.
close_all()	
get_manager_state([drop_defaults, widgets])	Returns the full state for a widget manager for embedding
get_state([key, drop_defaults])	Gets the widget state, or a piece of it.
get_view_spec()	
handle_comm_opened(comm, msg)	Static method, called when a widget is constructed.
has_trait(name)	Returns True if the object has a trait with the specified name.
hold_sync()	Hold syncing any state until the outermost context manager exits
hold_trait_notifications()	Context manager for bundling trait change notifications and cross validation.
notify_change(change)	Called when a property has changed.
observe(handler[, names, type])	Setup a handler to be called when a trait changes.
on_displayed(callback[, remove])	(Un)Register a widget displayed callback.
on_msg(callback[, remove])	(Un)Register a custom msg receive callback.
on_trait_change([handler, name, remove])	DEPRECATED: Setup a handler to be called when a trait changes.
on_widget_constructed(callback)	Registers a callback to be called when a widget is constructed.

Continued on next page

Table 29 – continued from previous page

open()	Open a comm to the frontend if one isn't already open.
send(content[, buffers])	Sends a custom msg to the widget model in the front-end.
send_state([key])	Sends the widget state, or a piece of it, to the front-end, if it exists.
set_state(sync_data)	Called when a state is received from the front-end.
set_trait(name, value)	Forcibly sets trait attribute, including read-only attributes.
setup_instance(*args, **kwargs)	This is called **before** self.__init__ is called.
trait_events([name])	Get a dict of all the event handlers of this class.
trait_metadata(traitname, key[, default])	Get metadata values for trait by key.
trait_names(**metadata)	Get a list of all the names of this class' traits.
traits(**metadata)	Get a dict of all the traits of this class.
unobserve(handler[, names, type])	Remove a trait change handler.
unobserve_all([name])	Remove trait change handlers of any type for the specified name.

Attributes

allow_padding	A boolean (True, False) trait.
center	An instance of a Python tuple.
clip_angle	A float trait.
comm	A trait whose value must be an instance of a specified class.
cross_validation_lock	A contextmanager for running a block with our cross validation lock set to True.
domain_class	A trait whose value must be a subclass of a specified class.
keys	An instance of a Python list.
log	A trait whose value must be an instance of a specified class.
model_id	Gets the model id of this widget.
precedence	
precision	A float trait.
reverse	A boolean (True, False) trait.
rotate	An instance of a Python tuple.
rtype	
scale_factor	A float trait.
scale_types	
widget_types	
widgets	

3.1.3 Marks

Mark(**kwargs)	The base mark class.
Lines(**kwargs)	Lines mark.
FlexLine(**kwargs)	Flexible Lines mark.
Scatter(**kwargs)	Scatter mark.

Continued on next page

Table 31 – continued from previous page

Hist(**kwargs)	Histogram mark.
Bars(**kwargs)	Bar mark.
Graph(**kwargs)	Graph with nodes and links.
GridHeatMap(**kwargs)	GridHeatMap mark.
HeatMap(**kwargs)	HeatMap mark.
Label(**kwargs)	Label mark.
OHLC(**kwargs)	Open/High/Low/Close marks.
Pie(**kwargs)	Piechart mark.
Map(**kwargs)	Map mark.

bqplot.marks.Mark

class bqplot.marks.**Mark**(***kwargs*)

The base mark class.

Traitlet mark attributes may be decorated with metadata.

Data Attribute Decoration

Data attributes are decorated with the following values:

scaled: bool Indicates whether the considered attribute is a data attribute which must be associated with a scale in order to be taken into account.

rtype: string Range type of the associated scale.

atype: string Key in bqplot's axis registry of the recommended axis type to represent this scale. When not specified, the default is 'bqplot.Axis'.

display_name

Holds a user-friendly name for the trait attribute.

Type string

mark_types

A registry of existing mark types.

Type dict (class-level attribute)

scales

A dictionary of scales holding scales for each data attribute. - If a mark holds a scaled attribute named 'x', the scales dictionary must have a corresponding scale for the key 'x'. - The scale's range type should be equal to the scaled attribute's range type (rtype).

Type Dict of scales (default: {})

scales_metadata

A dictionary of dictionaries holding metadata on the way scales are used by the mark. For example, a linear scale may be used to count pixels horizontally or vertically. The content of this dictionary may change dynamically. It is an instance-level attribute.

Type Dict (default: {})

preserve_domain

Indicates if this mark affects the domain(s) of the specified scale(s). The keys of this dictionary are the same as the ones of the "scales" attribute, and values are boolean. If a key is missing, it is considered as False.

Type dict (default: {})

display_legend
Display toggle for the mark legend in the general figure legend

Type bool (default: False)

labels
Labels of the items of the mark. This attribute has different meanings depending on the type of mark.

Type list of unicode strings (default: [])

apply_clip
Indicates whether the items that are beyond the limits of the chart should be clipped.

Type bool (default: True)

visible
Visibility toggle for the mark.

Type bool (default: True)

selected_style
CSS style to be applied to selected items in the mark.

Type dict (default: {})

unselected_style
CSS style to be applied to items that are not selected in the mark, when a selection exists.

Type dict (default: {})

selected
Indices of the selected items in the mark.

Type list of integers or None (default: None)

tooltip
Widget to be displayed as tooltip when elements of the scatter are hovered on

Type DOMWidget or None (default: None)

tooltip_style
Styles to be applied to the tooltip widget

Type Dictionary (default: {'opacity': 0.9})

enable_hover
Boolean attribute to control the hover interaction for the scatter. If this is false, the on_hover custom mssg is not sent back to the python side

Type Bool (default: True)

interactions
Dictionary listing the different interactions for each mark. The key is the event which triggers the interaction and the value is the kind of interactions. Keys and values can only take strings from separate enums for each mark.

Type Dictionary (default: {'hover': 'tooltip'})

tooltip_location
Enum specifying the location of the tooltip. 'mouse' places the tooltip at the location of the mouse when the tooltip is activated and 'center' places the tooltip at the center of the figure. If tooltip is linked to a click event, 'mouse' places the tooltip at the location of the click that triggered the tooltip to be visible.

Type {'mouse', 'center'} (default: 'mouse')

__init__(***kwargs*)
Public constructor

Methods

__init__(**kwargs)	Public constructor
add_traits(**traits)	Dynamically add trait attributes to the Widget.
class_own_trait_events(name)	Get a dict of all event handlers defined on this class, not a parent.
class_own_traits(**metadata)	Get a dict of all the traitlets defined on this class, not a parent.
class_trait_names(**metadata)	Get a list of all the names of this class' traits.
class_traits(**metadata)	Get a dict of all the traits of this class.
close()	Close method.
close_all()	
get_manager_state([drop_defaults, widgets])	Returns the full state for a widget manager for embedding
get_state([key, drop_defaults])	Gets the widget state, or a piece of it.
get_view_spec()	
handle_comm_opened(comm, msg)	Static method, called when a widget is constructed.
has_trait(name)	Returns True if the object has a trait with the specified name.
hold_sync()	Hold syncing any state until the outermost context manager exits
hold_trait_notifications()	Context manager for bundling trait change notifications and cross validation.
notify_change(change)	Called when a property has changed.
observe(handler[, names, type])	Setup a handler to be called when a trait changes.
on_background_click(callback[, remove])	
on_click(callback[, remove])	
on_displayed(callback[, remove])	(Un)Register a widget displayed callback.
on_element_click(callback[, remove])	
on_hover(callback[, remove])	
on_legend_click(callback[, remove])	
on_legend_hover(callback[, remove])	
on_msg(callback[, remove])	(Un)Register a custom msg receive callback.
on_trait_change([handler, name, remove])	DEPRECATED: Setup a handler to be called when a trait changes.
on_widget_constructed(callback)	Registers a callback to be called when a widget is constructed.
open()	Open a comm to the frontend if one isn't already open.
send(content[, buffers])	Sends a custom msg to the widget model in the front-end.
send_state([key])	Sends the widget state, or a piece of it, to the front-end, if it exists.
set_state(sync_data)	Called when a state is received from the front-end.
set_trait(name, value)	Forcibly sets trait attribute, including read-only attributes.
setup_instance(*args, **kwargs)	This is called **before** self.__init__ is called.

Continued on next page

Table 32 – continued from previous page

trait_events([name])	Get a dict of all the event handlers of this class.
trait_metadata(traitname, key[, default])	Get metadata values for trait by key.
trait_names(**metadata)	Get a list of all the names of this class' traits.
traits(**metadata)	Get a dict of all the traits of this class.
unobserve(handler[, names, type])	Remove a trait change handler.
unobserve_all([name])	Remove trait change handlers of any type for the specified name.

Attributes

apply_clip	A boolean (True, False) trait.
comm	A trait whose value must be an instance of a specified class.
cross_validation_lock	A contextmanager for running a block with our cross validation lock set to True.
display_legend	A boolean (True, False) trait.
enable_hover	A boolean (True, False) trait.
interactions	An instance of a Python dict.
keys	An instance of a Python list.
labels	An instance of a Python list.
log	A trait whose value must be an instance of a specified class.
mark_types	
model_id	Gets the model id of this widget.
preserve_domain	An instance of a Python dict.
scales	An instance of a Python dict.
scales_metadata	An instance of a Python dict.
selected	An instance of a Python list.
selected_style	An instance of a Python dict.
tooltip	A trait whose value must be an instance of a specified class.
tooltip_location	An enum whose value must be in a given sequence.
tooltip_style	An instance of a Python dict.
unselected_style	An instance of a Python dict.
visible	A boolean (True, False) trait.
widget_types	
widgets	

bqplot.marks.Lines

class bqplot.marks.**Lines**(***kwargs*)

Lines mark.

In the case of the Lines mark, scales for 'x' and 'y' MUST be provided.

icon

Font-awesome icon for the respective mark

Type string (class-level attribute)

name

User-friendly name of the mark

Type string (class-level attribute)

colors
List of colors of the Lines. If the list is shorter than the number of lines, the colors are reused.

Type list of colors (default: CATEGORY10)

close_path
Whether to close the paths or not.

Type bool (default: False)

fill
Fill in the area defined by the curves

Type {'none', 'bottom', 'top', 'inside', 'between'}

fill_colors
Fill colors for the areas. Defaults to stroke-colors when no color provided

Type list of colors (default: [])

opacities
Opacity for the lines and patches. Defaults to 1 when the list is too short, or the element of the list is set to None.

Type list of floats (default: [])

fill_opacities
Opacity for the areas. Defaults to 1 when the list is too short, or the element of the list is set to None.

Type list of floats (default: [])

stroke_width
Stroke width of the Lines

Type float (default: 2)

labels_visibility
Visibility of the curve labels

Type {'none', 'label'}

curves_subset
If set to None, all the lines are displayed. Otherwise, only the items in the list will have full opacity, while others will be faded.

Type list of integers or None (default: [])

line_style
Line style.

Type {'solid', 'dashed', 'dotted', 'dash_dotted'}

interpolation
Interpolation scheme used for interpolation between the data points provided. Please refer to the svg interpolate documentation for details about the different interpolation schemes.

Type {'linear', 'basis', 'cardinal', 'monotone'}

marker

'triangle-up', 'arrow', 'rectangle', 'ellipse'}

Marker shape

Type {'circle', 'cross', 'diamond', 'square', 'triangle-down',

marker_size
Default marker size in pixels

Type nonnegative int (default: 64)

Data Attributes

x
abscissas of the data points (1d or 2d array)

Type numpy.ndarray (default: [])

y
ordinates of the data points (1d or 2d array)

Type numpy.ndarray (default: [])

color
colors of the different lines based on data. If it is [], then the colors from the colors attribute are used. Each line has a single color and if the size of colors is less than the number of lines, the remaining lines are given the default colors.

Type numpy.ndarray (default: None)

Notes

The fields which can be passed to the default tooltip are: name: label of the line index: index of the line being hovered on color: data attribute for the color of the line

The following are the events which can trigger interactions: click: left click of the mouse hover: mouse-over an element

The following are the interactions which can be linked to the above events: tooltip: display tooltip

__init__(***kwargs*)
Public constructor

Methods

__init__(**kwargs)	Public constructor
add_traits(**traits)	Dynamically add trait attributes to the Widget.
class_own_trait_events(name)	Get a dict of all event handlers defined on this class, not a parent.
class_own_traits(**metadata)	Get a dict of all the traitlets defined on this class, not a parent.
class_trait_names(**metadata)	Get a list of all the names of this class' traits.
class_traits(**metadata)	Get a dict of all the traits of this class.
close()	Close method.
close_all()	
get_manager_state([drop_defaults, widgets])	Returns the full state for a widget manager for embedding
get_state([key, drop_defaults])	Gets the widget state, or a piece of it.
get_view_spec()	
handle_comm_opened(comm, msg)	Static method, called when a widget is constructed.

Continued on next page

Table 34 – continued from previous page

has_trait(name)	Returns True if the object has a trait with the specified name.
hold_sync()	Hold syncing any state until the outermost context manager exits
hold_trait_notifications()	Context manager for bundling trait change notifications and cross validation.
notify_change(change)	Called when a property has changed.
observe(handler[, names, type])	Setup a handler to be called when a trait changes.
on_background_click(callback[, remove])	
on_click(callback[, remove])	
on_displayed(callback[, remove])	(Un)Register a widget displayed callback.
on_element_click(callback[, remove])	
on_hover(callback[, remove])	
on_legend_click(callback[, remove])	
on_legend_hover(callback[, remove])	
on_msg(callback[, remove])	(Un)Register a custom msg receive callback.
on_trait_change([handler, name, remove])	DEPRECATED: Setup a handler to be called when a trait changes.
on_widget_constructed(callback)	Registers a callback to be called when a widget is constructed.
open()	Open a comm to the frontend if one isn't already open.
send(content[, buffers])	Sends a custom msg to the widget model in the front-end.
send_state([key])	Sends the widget state, or a piece of it, to the front-end, if it exists.
set_state(sync_data)	Called when a state is received from the front-end.
set_trait(name, value)	Forcibly sets trait attribute, including read-only attributes.
setup_instance(*args, **kwargs)	This is called **before** self.__init__ is called.
trait_events([name])	Get a dict of all the event handlers of this class.
trait_metadata(traitname, key[, default])	Get metadata values for trait by key.
trait_names(**metadata)	Get a list of all the names of this class' traits.
traits(**metadata)	Get a dict of all the traits of this class.
unobserve(handler[, names, type])	Remove a trait change handler.
unobserve_all([name])	Remove trait change handlers of any type for the specified name.

Attributes

apply_clip	A boolean (True, False) trait.
close_path	A boolean (True, False) trait.
color	A numpy array trait type.
colors	An instance of a Python list.
comm	A trait whose value must be an instance of a specified class.
cross_validation_lock	A contextmanager for running a block with our cross validation lock set to True.
curves_subset	An instance of a Python list.
display_legend	A boolean (True, False) trait.

Continued on next page

Table 35 – continued from previous page

enable_hover	A boolean (True, False) trait.
fill	An enum whose value must be in a given sequence.
fill_colors	An instance of a Python list.
fill_opacities	An instance of a Python list.
icon	
interactions	An instance of a Python dict.
interpolation	An enum whose value must be in a given sequence.
keys	An instance of a Python list.
labels	An instance of a Python list.
labels_visibility	An enum whose value must be in a given sequence.
line_style	An enum whose value must be in a given sequence.
log	A trait whose value must be an instance of a specified class.
mark_types	
marker	An enum whose value must be in a given sequence.
marker_size	An int trait.
model_id	Gets the model id of this widget.
name	
opacities	An instance of a Python list.
preserve_domain	An instance of a Python dict.
scales	An instance of a Python dict.
scales_metadata	An instance of a Python dict.
selected	An instance of a Python list.
selected_style	An instance of a Python dict.
stroke_width	A float trait.
tooltip	A trait whose value must be an instance of a specified class.
tooltip_location	An enum whose value must be in a given sequence.
tooltip_style	An instance of a Python dict.
unselected_style	An instance of a Python dict.
visible	A boolean (True, False) trait.
widget_types	
widgets	
x	A numpy array trait type.
y	A numpy array trait type.

bqplot.marks.FlexLine

class bqplot.marks.**FlexLine**(***kwargs*)

Flexible Lines mark.

In the case of the FlexLines mark, scales for 'x' and 'y' MUST be provided. Scales for the color and width data attributes are optional. In the case where another data attribute than 'x' or 'y' is provided but the corresponding scale is missing, the data attribute is ignored.

name

user-friendly name of the mark

Type string (class-level attributes)

colors

List of colors for the Lines

Type list of colors (default: CATEGORY10)

stroke_width
Default stroke width of the Lines

Type float (default: 1.5)

Data Attributes

x
abscissas of the data points (1d array)

Type numpy.ndarray (default: [])

y
ordinates of the data points (1d array)

Type numpy.ndarray (default: [])

color
Array controlling the color of the data points

Type numpy.ndarray or None (default: None)

width
Array controlling the widths of the Lines.

Type numpy.ndarray or None (default: None)

__init__(***kwargs*)
Public constructor

Methods

__init__(**kwargs)	Public constructor
add_traits(**traits)	Dynamically add trait attributes to the Widget.
class_own_trait_events(name)	Get a dict of all event handlers defined on this class, not a parent.
class_own_traits(**metadata)	Get a dict of all the traitlets defined on this class, not a parent.
class_trait_names(**metadata)	Get a list of all the names of this class' traits.
class_traits(**metadata)	Get a dict of all the traits of this class.
close()	Close method.
close_all()	
get_manager_state([drop_defaults, widgets])	Returns the full state for a widget manager for embedding
get_state([key, drop_defaults])	Gets the widget state, or a piece of it.
get_view_spec()	
handle_comm_opened(comm, msg)	Static method, called when a widget is constructed.
has_trait(name)	Returns True if the object has a trait with the specified name.
hold_sync()	Hold syncing any state until the outermost context manager exits
hold_trait_notifications()	Context manager for bundling trait change notifications and cross validation.
notify_change(change)	Called when a property has changed.
observe(handler[, names, type])	Setup a handler to be called when a trait changes.
on_background_click(callback[, remove])	

Continued on next page

Table 36 – continued from previous page

on_click(callback[, remove])	
on_displayed(callback[, remove])	(Un)Register a widget displayed callback.
on_element_click(callback[, remove])	
on_hover(callback[, remove])	
on_legend_click(callback[, remove])	
on_legend_hover(callback[, remove])	
on_msg(callback[, remove])	(Un)Register a custom msg receive callback.
on_trait_change([handler, name, remove])	DEPRECATED: Setup a handler to be called when a trait changes.
on_widget_constructed(callback)	Registers a callback to be called when a widget is constructed.
open()	Open a comm to the frontend if one isn't already open.
send(content[, buffers])	Sends a custom msg to the widget model in the front-end.
send_state([key])	Sends the widget state, or a piece of it, to the front-end, if it exists.
set_state(sync_data)	Called when a state is received from the front-end.
set_trait(name, value)	Forcibly sets trait attribute, including read-only attributes.
setup_instance(*args, **kwargs)	This is called **before** self.__init__ is called.
trait_events([name])	Get a dict of all the event handlers of this class.
trait_metadata(traitname, key[, default])	Get metadata values for trait by key.
trait_names(**metadata)	Get a list of all the names of this class' traits.
traits(**metadata)	Get a dict of all the traits of this class.
unobserve(handler[, names, type])	Remove a trait change handler.
unobserve_all([name])	Remove trait change handlers of any type for the specified name.

Attributes

apply_clip	A boolean (True, False) trait.
color	A numpy array trait type.
colors	An instance of a Python list.
comm	A trait whose value must be an instance of a specified class.
cross_validation_lock	A contextmanager for running a block with our cross validation lock set to True.
display_legend	A boolean (True, False) trait.
enable_hover	A boolean (True, False) trait.
icon	
interactions	An instance of a Python dict.
keys	An instance of a Python list.
labels	An instance of a Python list.
log	A trait whose value must be an instance of a specified class.
mark_types	
model_id	Gets the model id of this widget.
name	
preserve_domain	An instance of a Python dict.

Continued on next page

Table 37 – continued from previous page

scales	An instance of a Python dict.
scales_metadata	An instance of a Python dict.
selected	An instance of a Python list.
selected_style	An instance of a Python dict.
stroke_width	A float trait.
tooltip	A trait whose value must be an instance of a specified class.
tooltip_location	An enum whose value must be in a given sequence.
tooltip_style	An instance of a Python dict.
unselected_style	An instance of a Python dict.
visible	A boolean (True, False) trait.
widget_types	
widgets	
width	A numpy array trait type.
x	A numpy array trait type.
y	A numpy array trait type.

bqplot.marks.Scatter

class bqplot.marks.**Scatter**(***kwargs*)

Scatter mark.

In the case of the Scatter mark, scales for 'x' and 'y' MUST be provided. The scales of other data attributes are optional. In the case where another data attribute than 'x' or 'y' is provided but the corresponding scale is missing, the data attribute is ignored.

icon

Font-awesome icon for that mark

Type string (class-level attribute)

name

User-friendly name of the mark

Type string (class-level attribute)

marker

'triangle-up', 'arrow', 'rectangle', 'ellipse'}

Marker shape

Type {'circle', 'cross', 'diamond', 'square', 'triangle-down',

colors

List of colors of the markers. If the list is shorter than the number of points, the colors are reused.

Type list of colors (default: ['steelblue'])

default_colors

Same as *colors*, deprecated as of version 0.8.4

Type Deprecated

stroke

Stroke color of the marker

Type Color or None (default: None)

stroke_width
Stroke width of the marker

Type Float (default: 1.5)

default_opacities
Default opacities of the markers. If the list is shorter than the number of points, the opacities are reused.

Type list of floats (default: [1.0])

default_skew
Default skew of the marker. This number is validated to be between 0 and 1.

Type float (default: 0.5)

default_size
Default marker size in pixel. If size data is provided with a scale, default_size stands for the maximal marker size (i.e. the maximum value for the 'size' scale range)

Type nonnegative int (default: 64)

drag_size
Ratio of the size of the dragged scatter size to the default scatter size.

Type nonnegative float (default: 5.)

names
Labels for the points of the chart

Type numpy.ndarray (default: None)

display_names
Controls whether names are displayed for points in the scatter

Type bool (default: True)

enable_move
Controls whether points can be moved by dragging. Refer to restrict_x, restrict_y for more options.

Type bool (default: False)

restrict_x
Restricts movement of the point to only along the x axis. This is valid only when enable_move is set to True. If both restrict_x and restrict_y are set to True, the point cannot be moved.

Type bool (default: False)

restrict_y
Restricts movement of the point to only along the y axis. This is valid only when enable_move is set to True. If both restrict_x and restrict_y are set to True, the point cannot be moved.

Type bool (default: False)

Data Attributes

x: numpy.ndarray (default: []) abscissas of the data points (1d array)

y: numpy.ndarray (default: []) ordinates of the data points (1d array)

color: numpy.ndarray or None (default: None) color of the data points (1d array). Defaults to default_color when not provided or when a value is NaN

opacity: numpy.ndarray or None (default: None) opacity of the data points (1d array). Defaults to default_opacity when not provided or when a value is NaN

size: numpy.ndarray or None (default: None) size of the data points. Defaults to default_size when not provided or when a value is NaN

skew: numpy.ndarray or None (default: None) skewness of the markers representing the data points. Defaults to default_skew when not provided or when a value is NaN

rotation: numpy.ndarray or None (default: None) orientation of the markers representing the data points. The rotation scale's range is [0, 180] Defaults to 0 when not provided or when a value is NaN.

Notes

The fields which can be passed to the default tooltip are: All the data attributes index: index of the marker being hovered on

The following are the events which can trigger interactions: click: left click of the mouse hover: mouse-over an element

The following are the interactions which can be linked to the above events: tooltip: display tooltip add: add new points to the scatter (can only linked to click)

__init__(***kwargs*)
Public constructor

Methods

__init__(**kwargs)	Public constructor
add_traits(**traits)	Dynamically add trait attributes to the Widget.
class_own_trait_events(name)	Get a dict of all event handlers defined on this class, not a parent.
class_own_traits(**metadata)	Get a dict of all the traitlets defined on this class, not a parent.
class_trait_names(**metadata)	Get a list of all the names of this class' traits.
class_traits(**metadata)	Get a dict of all the traits of this class.
close()	Close method.
close_all()	
get_manager_state([drop_defaults, widgets])	Returns the full state for a widget manager for embedding
get_state([key, drop_defaults])	Gets the widget state, or a piece of it.
get_view_spec()	
handle_comm_opened(comm, msg)	Static method, called when a widget is constructed.
has_trait(name)	Returns True if the object has a trait with the specified name.
hold_sync()	Hold syncing any state until the outermost context manager exits
hold_trait_notifications()	Context manager for bundling trait change notifications and cross validation.
notify_change(change)	Called when a property has changed.
observe(handler[, names, type])	Setup a handler to be called when a trait changes.
on_background_click(callback[, remove])	
on_click(callback[, remove])	
on_displayed(callback[, remove])	(Un)Register a widget displayed callback.
on_drag(callback[, remove])	

Continued on next page

Table 38 – continued from previous page

on_drag_end(callback[, remove])	
on_drag_start(callback[, remove])	
on_element_click(callback[, remove])	
on_hover(callback[, remove])	
on_legend_click(callback[, remove])	
on_legend_hover(callback[, remove])	
on_msg(callback[, remove])	(Un)Register a custom msg receive callback.
on_trait_change([handler, name, remove])	DEPRECATED: Setup a handler to be called when a trait changes.
on_widget_constructed(callback)	Registers a callback to be called when a widget is constructed.
open()	Open a comm to the frontend if one isn't already open.
send(content[, buffers])	Sends a custom msg to the widget model in the front-end.
send_state([key])	Sends the widget state, or a piece of it, to the front-end, if it exists.
set_state(sync_data)	Called when a state is received from the front-end.
set_trait(name, value)	Forcibly sets trait attribute, including read-only attributes.
setup_instance(*args, **kwargs)	This is called **before** self.__init__ is called.
trait_events([name])	Get a dict of all the event handlers of this class.
trait_metadata(traitname, key[, default])	Get metadata values for trait by key.
trait_names(**metadata)	Get a list of all the names of this class' traits.
traits(**metadata)	Get a dict of all the traits of this class.
unobserve(handler[, names, type])	Remove a trait change handler.
unobserve_all([name])	Remove trait change handlers of any type for the specified name.

Attributes

apply_clip	A boolean (True, False) trait.
color	A numpy array trait type.
colors	An instance of a Python list.
comm	A trait whose value must be an instance of a specified class.
cross_validation_lock	A contextmanager for running a block with our cross validation lock set to True.
default_colors	
default_opacities	An instance of a Python list.
default_size	An int trait.
default_skew	A float trait.
display_legend	A boolean (True, False) trait.
display_names	A boolean (True, False) trait.
drag_color	A string holding a valid HTML color such as 'blue', '#060482', '#A80'
drag_size	A float trait.
enable_delete	A boolean (True, False) trait.
enable_hover	A boolean (True, False) trait.
enable_move	A boolean (True, False) trait.

Continued on next page

Table 39 – continued from previous page

fill	A boolean (True, False) trait.
hovered_point	An int trait.
hovered_style	An instance of a Python dict.
icon	
interactions	An instance of a Python dict.
keys	An instance of a Python list.
labels	An instance of a Python list.
log	A trait whose value must be an instance of a specified class.
mark_types	
marker	An enum whose value must be in a given sequence.
model_id	Gets the model id of this widget.
name	
names	A numpy array trait type.
names_unique	A boolean (True, False) trait.
opacity	A numpy array trait type.
preserve_domain	An instance of a Python dict.
restrict_x	A boolean (True, False) trait.
restrict_y	A boolean (True, False) trait.
rotation	A numpy array trait type.
scales	An instance of a Python dict.
scales_metadata	An instance of a Python dict.
selected	An instance of a Python list.
selected_style	An instance of a Python dict.
size	A numpy array trait type.
skew	A numpy array trait type.
stroke	A string holding a valid HTML color such as 'blue', '#060482', '#A80'
stroke_width	A float trait.
tooltip	A trait whose value must be an instance of a specified class.
tooltip_location	An enum whose value must be in a given sequence.
tooltip_style	An instance of a Python dict.
unhovered_style	An instance of a Python dict.
unselected_style	An instance of a Python dict.
update_on_move	A boolean (True, False) trait.
visible	A boolean (True, False) trait.
widget_types	
widgets	
x	A numpy array trait type.
y	A numpy array trait type.

bqplot.marks.Hist

class bqplot.marks.**Hist**(***kwargs*)

Histogram mark.

In the case of the Hist mark, scales for 'sample' and 'count' MUST be provided.

icon

font-awesome icon for that mark

Type string (class-level attribute)

name
user-friendly name of the mark

Type string (class-level attribute)

bins
number of bins in the histogram

Type nonnegative int (default: 10)

normalized
Boolean attribute to return normalized values which sum to 1 or direct counts for the *count* attribute. The scale of *count* attribute is determined by the value of this flag.

Type bool (default: False)

colors
List of colors of the Histogram. If the list is shorter than the number of bins, the colors are reused.

Type list of colors (default: CATEGORY10)

stroke
Stroke color of the histogram

Type Color or None (default: None)

opacities
Opacity for the bins of the histogram. Defaults to 1 when the list is too short, or the element of the list is set to None.

Type list of floats (default: [])

midpoints
midpoints of the bins of the histogram. It is a read-only attribute.

Type list (default: [])

Data Attributes

sample
sample of which the histogram must be computed.

Type numpy.ndarray (default: [])

count
number of sample points per bin. It is a read-only attribute.

Type numpy.ndarray (read-only)

Notes

The fields which can be passed to the default tooltip are: midpoint: mid-point of the bin related to the rectangle hovered on count: number of elements in the bin hovered on bin_start: start point of the bin bin-end: end point of the bin index: index of the bin

__init__ (***kwargs*)
Public constructor

Methods

__init__(**kwargs)	Public constructor
add_traits(**traits)	Dynamically add trait attributes to the Widget.
class_own_trait_events(name)	Get a dict of all event handlers defined on this class, not a parent.
class_own_traits(**metadata)	Get a dict of all the traitlets defined on this class, not a parent.
class_trait_names(**metadata)	Get a list of all the names of this class' traits.
class_traits(**metadata)	Get a dict of all the traits of this class.
close()	Close method.
close_all()	
get_manager_state([drop_defaults, widgets])	Returns the full state for a widget manager for embedding
get_state([key, drop_defaults])	Gets the widget state, or a piece of it.
get_view_spec()	
handle_comm_opened(comm, msg)	Static method, called when a widget is constructed.
has_trait(name)	Returns True if the object has a trait with the specified name.
hold_sync()	Hold syncing any state until the outermost context manager exits
hold_trait_notifications()	Context manager for bundling trait change notifications and cross validation.
notify_change(change)	Called when a property has changed.
observe(handler[, names, type])	Setup a handler to be called when a trait changes.
on_background_click(callback[, remove])	
on_click(callback[, remove])	
on_displayed(callback[, remove])	(Un)Register a widget displayed callback.
on_element_click(callback[, remove])	
on_hover(callback[, remove])	
on_legend_click(callback[, remove])	
on_legend_hover(callback[, remove])	
on_msg(callback[, remove])	(Un)Register a custom msg receive callback.
on_trait_change([handler, name, remove])	DEPRECATED: Setup a handler to be called when a trait changes.
on_widget_constructed(callback)	Registers a callback to be called when a widget is constructed.
open()	Open a comm to the frontend if one isn't already open.
send(content[, buffers])	Sends a custom msg to the widget model in the front-end.
send_state([key])	Sends the widget state, or a piece of it, to the front-end, if it exists.
set_state(sync_data)	Called when a state is received from the front-end.
set_trait(name, value)	Forcibly sets trait attribute, including read-only attributes.
setup_instance(*args, **kwargs)	This is called **before** self.__init__ is called.
trait_events([name])	Get a dict of all the event handlers of this class.
trait_metadata(traitname, key[, default])	Get metadata values for trait by key.
trait_names(**metadata)	Get a list of all the names of this class' traits.
traits(**metadata)	Get a dict of all the traits of this class.
unobserve(handler[, names, type])	Remove a trait change handler.

Continued on next page

Table 40 – continued from previous page

unobserve_all([name])	Remove trait change handlers of any type for the specified name.

Attributes

apply_clip	A boolean (True, False) trait.
bins	An int trait.
colors	An instance of a Python list.
comm	A trait whose value must be an instance of a specified class.
count	A numpy array trait type.
cross_validation_lock	A contextmanager for running a block with our cross validation lock set to True.
display_legend	A boolean (True, False) trait.
enable_hover	A boolean (True, False) trait.
icon	
interactions	An instance of a Python dict.
keys	An instance of a Python list.
labels	An instance of a Python list.
log	A trait whose value must be an instance of a specified class.
mark_types	
midpoints	An instance of a Python list.
model_id	Gets the model id of this widget.
name	
normalized	A boolean (True, False) trait.
opacities	An instance of a Python list.
preserve_domain	An instance of a Python dict.
sample	A numpy array trait type.
scales	An instance of a Python dict.
scales_metadata	An instance of a Python dict.
selected	An instance of a Python list.
selected_style	An instance of a Python dict.
stroke	A string holding a valid HTML color such as 'blue', '#060482', '#A80'
tooltip	A trait whose value must be an instance of a specified class.
tooltip_location	An enum whose value must be in a given sequence.
tooltip_style	An instance of a Python dict.
unselected_style	An instance of a Python dict.
visible	A boolean (True, False) trait.
widget_types	
widgets	

bqplot.marks.Bars

class bqplot.marks.**Bars**(***kwargs*)

Bar mark.

In the case of the Bars mark, scales for 'x' and 'y' MUST be provided. The scales of other data attributes are optional. In the case where another data attribute than 'x' or 'y' is provided but the corresponding scale is

missing, the data attribute is ignored.

icon
font-awesome icon for that mark

Type string (class-level attribute)

name
user-friendly name of the mark

Type string (class-level attribute)

color_mode
enum attribute to specify if color should be the same for all bars with the same x or for all bars which belong to the same array in Y 'group' means for every x all bars have same color. 'element' means for every dimension of y, all bars have same color. 'auto' picks 'group' and 'element' for 1-d and 2-d values of Y respectively.

Type {'auto', 'group', 'element'}

type
whether 2-dimensional bar charts should appear grouped or stacked.

Type {'stacked', 'grouped'}

colors
list of colors for the bars.

Type list of colors (default: ['steelblue'])

orientation
Specifies whether the bar chart is drawn horizontally or vertically. If a horizontal bar chart is drawn, the x data is drawn vertically.

Type {'horizontal', 'vertical'}

padding
attribute to control the spacing between the bars value is specified as a percentage of the width of the bar

Type float (default: 0.05)

stroke
stroke color for the bars

Type Color or None (default: None)

opacities
Opacities for the bars. Defaults to 1 when the list is too short, or the element of the list is set to None.

Type list of floats (default: [])

base
reference value from which the bars are drawn. defaults to 0.0

Type float (default: 0.0)

align
alignment of bars with respect to the tick value

Type {'center', 'left', 'right'}

Data Attributes

x
abscissas of the data points (1d array)

Type numpy.ndarray (default: [])

y
ordinates of the values for the data points

Type numpy.ndarray (default: [])

color
color of the data points (1d array). Defaults to default_color when not provided or when a value is NaN

Type numpy.ndarray or None (default: None)

Notes

The fields which can be passed to the default tooltip are: All the data attributes index: index of the bar being hovered on sub_index: if data is two dimensional, this is the minor index

__init__(***kwargs*)
Public constructor

Methods

__init__(**kwargs)	Public constructor
add_traits(**traits)	Dynamically add trait attributes to the Widget.
class_own_trait_events(name)	Get a dict of all event handlers defined on this class, not a parent.
class_own_traits(**metadata)	Get a dict of all the traitlets defined on this class, not a parent.
class_trait_names(**metadata)	Get a list of all the names of this class' traits.
class_traits(**metadata)	Get a dict of all the traits of this class.
close()	Close method.
close_all()	
get_manager_state([drop_defaults, widgets])	Returns the full state for a widget manager for embedding
get_state([key, drop_defaults])	Gets the widget state, or a piece of it.
get_view_spec()	
handle_comm_opened(comm, msg)	Static method, called when a widget is constructed.
has_trait(name)	Returns True if the object has a trait with the specified name.
hold_sync()	Hold syncing any state until the outermost context manager exits
hold_trait_notifications()	Context manager for bundling trait change notifications and cross validation.
notify_change(change)	Called when a property has changed.
observe(handler[, names, type])	Setup a handler to be called when a trait changes.
on_background_click(callback[, remove])	
on_click(callback[, remove])	
on_displayed(callback[, remove])	(Un)Register a widget displayed callback.
on_element_click(callback[, remove])	
on_hover(callback[, remove])	
on_legend_click(callback[, remove])	
on_legend_hover(callback[, remove])	

Continued on next page

Table 42 – continued from previous page

on_msg(callback[, remove])	(Un)Register a custom msg receive callback.
on_trait_change([handler, name, remove])	DEPRECATED: Setup a handler to be called when a trait changes.
on_widget_constructed(callback)	Registers a callback to be called when a widget is constructed.
open()	Open a comm to the frontend if one isn't already open.
send(content[, buffers])	Sends a custom msg to the widget model in the front-end.
send_state([key])	Sends the widget state, or a piece of it, to the front-end, if it exists.
set_state(sync_data)	Called when a state is received from the front-end.
set_trait(name, value)	Forcibly sets trait attribute, including read-only attributes.
setup_instance(*args, **kwargs)	This is called **before** self.__init__ is called.
trait_events([name])	Get a dict of all the event handlers of this class.
trait_metadata(traitname, key[, default])	Get metadata values for trait by key.
trait_names(**metadata)	Get a list of all the names of this class' traits.
traits(**metadata)	Get a dict of all the traits of this class.
unobserve(handler[, names, type])	Remove a trait change handler.
unobserve_all([name])	Remove trait change handlers of any type for the specified name.

Attributes

align	An enum whose value must be in a given sequence.
apply_clip	A boolean (True, False) trait.
base	A float trait.
color	A numpy array trait type.
color_mode	An enum whose value must be in a given sequence.
colors	An instance of a Python list.
comm	A trait whose value must be an instance of a specified class.
cross_validation_lock	A contextmanager for running a block with our cross validation lock set to True.
display_legend	A boolean (True, False) trait.
enable_hover	A boolean (True, False) trait.
icon	
interactions	An instance of a Python dict.
keys	An instance of a Python list.
labels	An instance of a Python list.
log	A trait whose value must be an instance of a specified class.
mark_types	
model_id	Gets the model id of this widget.
name	
opacities	An instance of a Python list.
orientation	An enum whose value must be in a given sequence.
padding	A float trait.
preserve_domain	An instance of a Python dict.

Continued on next page

Table 43 – continued from previous page

scales	An instance of a Python dict.
scales_metadata	An instance of a Python dict.
selected	An instance of a Python list.
selected_style	An instance of a Python dict.
stroke	A string holding a valid HTML color such as 'blue', '#060482', '#A80'
tooltip	A trait whose value must be an instance of a specified class.
tooltip_location	An enum whose value must be in a given sequence.
tooltip_style	An instance of a Python dict.
type	An enum whose value must be in a given sequence.
unselected_style	An instance of a Python dict.
visible	A boolean (True, False) trait.
widget_types	
widgets	
x	A numpy array trait type.
y	A numpy array trait type.

bqplot.marks.Graph

class bqplot.marks.**Graph**(***kwargs*)

Graph with nodes and links.

node_data

list of node attributes for the graph

Type List

link_matrix

link data passed as 2d matrix

Type numpy.ndarray of shape(len(nodes), len(nodes))

link_data

list of link attributes for the graph

Type List

charge

charge of force layout. Will be ignored when x and y data attributes are set

Type int (default: -300)

link_distance

link distance in pixels between nodes. Will be ignored when x and y data attributes are set

Type float (default: 100)

link_type

Enum representing link type

Type {'arc', 'line', 'slant_line'} (default: 'arc')

directed

directed or undirected graph

Type bool (default: True)

highlight_links

highlights incoming and outgoing links when hovered on a node

Type bool (default: True)

colors
list of node colors

Type list (default: CATEGORY10)

Data Attributes

x
abscissas of the node data points (1d array)

Type numpy.ndarray (default: [])

y
ordinates of the node data points (1d array)

Type numpy.ndarray (default: [])

color
color of the node data points (1d array).

Type numpy.ndarray or None (default: None)

link_color
link data passed as 2d matrix

Type numpy.ndarray of shape(len(nodes), len(nodes))

__init__(***kwargs*)
Public constructor

Methods

__init__(**kwargs)	Public constructor
add_traits(**traits)	Dynamically add trait attributes to the Widget.
class_own_trait_events(name)	Get a dict of all event handlers defined on this class, not a parent.
class_own_traits(**metadata)	Get a dict of all the traitlets defined on this class, not a parent.
class_trait_names(**metadata)	Get a list of all the names of this class' traits.
class_traits(**metadata)	Get a dict of all the traits of this class.
close()	Close method.
close_all()	
get_manager_state([drop_defaults, widgets])	Returns the full state for a widget manager for embedding
get_state([key, drop_defaults])	Gets the widget state, or a piece of it.
get_view_spec()	
handle_comm_opened(comm, msg)	Static method, called when a widget is constructed.
has_trait(name)	Returns True if the object has a trait with the specified name.
hold_sync()	Hold syncing any state until the outermost context manager exits
hold_trait_notifications()	Context manager for bundling trait change notifications and cross validation.
notify_change(change)	Called when a property has changed.
observe(handler[, names, type])	Setup a handler to be called when a trait changes.

Continued on next page

Table 44 – continued from previous page

on_background_click(callback[, remove])	
on_click(callback[, remove])	
on_displayed(callback[, remove])	(Un)Register a widget displayed callback.
on_element_click(callback[, remove])	
on_hover(callback[, remove])	
on_legend_click(callback[, remove])	
on_legend_hover(callback[, remove])	
on_msg(callback[, remove])	(Un)Register a custom msg receive callback.
on_trait_change([handler, name, remove])	DEPRECATED: Setup a handler to be called when a trait changes.
on_widget_constructed(callback)	Registers a callback to be called when a widget is constructed.
open()	Open a comm to the frontend if one isn't already open.
send(content[, buffers])	Sends a custom msg to the widget model in the front-end.
send_state([key])	Sends the widget state, or a piece of it, to the front-end, if it exists.
set_state(sync_data)	Called when a state is received from the front-end.
set_trait(name, value)	Forcibly sets trait attribute, including read-only attributes.
setup_instance(*args, **kwargs)	This is called **before** self.__init__ is called.
trait_events([name])	Get a dict of all the event handlers of this class.
trait_metadata(traitname, key[, default])	Get metadata values for trait by key.
trait_names(**metadata)	Get a list of all the names of this class' traits.
traits(**metadata)	Get a dict of all the traits of this class.
unobserve(handler[, names, type])	Remove a trait change handler.
unobserve_all([name])	Remove trait change handlers of any type for the specified name.

Attributes

apply_clip	A boolean (True, False) trait.
charge	An int trait.
color	A numpy array trait type.
colors	An instance of a Python list.
comm	A trait whose value must be an instance of a specified class.
cross_validation_lock	A contextmanager for running a block with our cross validation lock set to True.
directed	A boolean (True, False) trait.
display_legend	A boolean (True, False) trait.
enable_hover	A boolean (True, False) trait.
highlight_links	A boolean (True, False) trait.
hovered_point	An int trait.
hovered_style	An instance of a Python dict.
interactions	An instance of a Python dict.
keys	An instance of a Python list.
labels	An instance of a Python list.
link_color	A numpy array trait type.

Continued on next page

Table 45 – continued from previous page

link_data	An instance of a Python list.
link_distance	A float trait.
link_matrix	A numpy array trait type.
link_type	An enum whose value must be in a given sequence.
log	A trait whose value must be an instance of a specified class.
mark_types	
model_id	Gets the model id of this widget.
node_data	An instance of a Python list.
preserve_domain	An instance of a Python dict.
scales	An instance of a Python dict.
scales_metadata	An instance of a Python dict.
selected	An instance of a Python list.
selected_style	An instance of a Python dict.
tooltip	A trait whose value must be an instance of a specified class.
tooltip_location	An enum whose value must be in a given sequence.
tooltip_style	An instance of a Python dict.
unhovered_style	An instance of a Python dict.
unselected_style	An instance of a Python dict.
visible	A boolean (True, False) trait.
widget_types	
widgets	
x	A numpy array trait type.
y	A numpy array trait type.

bqplot.marks.GridHeatMap

class bqplot.marks.**GridHeatMap**(***kwargs*)

GridHeatMap mark.

Alignment: The tiles can be aligned so that the data matches either the start, the end or the midpoints of the tiles. This is controlled by the align attribute.

Suppose the data passed is a m-by-n matrix. If the scale for the rows is Ordinal, then alignment is by default the mid points. For a non-ordinal scale, the data cannot be aligned to the mid points of the rectangles.

If it is not ordinal, then two cases arise. If the number of rows passed is m, then align attribute can be used. If the number of rows passed is m+1, then the data are the boundaries of the m rectangles.

If rows and columns are not passed, and scales for them are also not passed, then ordinal scales are generated for the rows and columns.

row_align

This is only valid if the number of entries in *row* exactly match the number of rows in *color* and the *row_scale* is not *OrdinalScale*. *start* aligns the row values passed to be aligned with the start of the tiles and *end* aligns the row values to the end of the tiles.

Type Enum([‘start’, ‘end’])

column_align

This is only valid if the number of entries in *column* exactly match the number of columns in *color* and the *column_scale* is not *OrdinalScale*. *start* aligns the column values passed to be aligned with the start of the tiles and *end* aligns the column values to the end of the tiles.

Type Enum([‘start’, end’])

anchor_style
Controls the style for the element which serves as the anchor during selection.

Type dict (default: {'fill': 'white', 'stroke': 'blue'})

Data Attributes

color
color of the data points (2d array). The number of elements in this array correspond to the number of cells created in the heatmap.

Type numpy.ndarray or None (default: None)

row
labels for the rows of the *color* array passed. The length of this can be no more than 1 away from the number of rows in *color*. This is a scaled attribute and can be used to affect the height of the cells as the entries of *row* can indicate the start or the end points of the cells. Refer to the property *row_align*. If this property is None, then a uniformly spaced grid is generated in the row direction.

Type numpy.ndarray or None (default: None)

column
labels for the columns of the *color* array passed. The length of this can be no more than 1 away from the number of columns in *color* This is a scaled attribute and can be used to affect the width of the cells as the entries of *column* can indicate the start or the end points of the cells. Refer to the property *column_align*. If this property is None, then a uniformly spaced grid is generated in the column direction.

Type numpy.ndarray or None (default: None)

__init__(***kwargs*)
Public constructor

Methods

__init__(**kwargs)	Public constructor
add_traits(**traits)	Dynamically add trait attributes to the Widget.
class_own_trait_events(name)	Get a dict of all event handlers defined on this class, not a parent.
class_own_traits(**metadata)	Get a dict of all the traitlets defined on this class, not a parent.
class_trait_names(**metadata)	Get a list of all the names of this class' traits.
class_traits(**metadata)	Get a dict of all the traits of this class.
close()	Close method.
close_all()	
get_manager_state([drop_defaults, widgets])	Returns the full state for a widget manager for embedding
get_state([key, drop_defaults])	Gets the widget state, or a piece of it.
get_view_spec()	
handle_comm_opened(comm, msg)	Static method, called when a widget is constructed.
has_trait(name)	Returns True if the object has a trait with the specified name.
hold_sync()	Hold syncing any state until the outermost context manager exits
hold_trait_notifications()	Context manager for bundling trait change notifications and cross validation.

Continued on next page

Table 46 – continued from previous page

notify_change(change)	Called when a property has changed.
observe(handler[, names, type])	Setup a handler to be called when a trait changes.
on_background_click(callback[, remove])	
on_click(callback[, remove])	
on_displayed(callback[, remove])	(Un)Register a widget displayed callback.
on_element_click(callback[, remove])	
on_hover(callback[, remove])	
on_legend_click(callback[, remove])	
on_legend_hover(callback[, remove])	
on_msg(callback[, remove])	(Un)Register a custom msg receive callback.
on_trait_change([handler, name, remove])	DEPRECATED: Setup a handler to be called when a trait changes.
on_widget_constructed(callback)	Registers a callback to be called when a widget is constructed.
open()	Open a comm to the frontend if one isn't already open.
send(content[, buffers])	Sends a custom msg to the widget model in the front-end.
send_state([key])	Sends the widget state, or a piece of it, to the front-end, if it exists.
set_state(sync_data)	Called when a state is received from the front-end.
set_trait(name, value)	Forcibly sets trait attribute, including read-only attributes.
setup_instance(*args, **kwargs)	This is called **before** self.__init__ is called.
trait_events([name])	Get a dict of all the event handlers of this class.
trait_metadata(traitname, key[, default])	Get metadata values for trait by key.
trait_names(**metadata)	Get a list of all the names of this class' traits.
traits(**metadata)	Get a dict of all the traits of this class.
unobserve(handler[, names, type])	Remove a trait change handler.
unobserve_all([name])	Remove trait change handlers of any type for the specified name.

Attributes

anchor_style	An instance of a Python dict.
apply_clip	A boolean (True, False) trait.
color	A numpy array trait type.
column	A numpy array trait type.
column_align	An enum whose value must be in a given sequence.
comm	A trait whose value must be an instance of a specified class.
cross_validation_lock	A contextmanager for running a block with our cross validation lock set to True.
display_legend	A boolean (True, False) trait.
enable_hover	A boolean (True, False) trait.
interactions	An instance of a Python dict.
keys	An instance of a Python list.
labels	An instance of a Python list.
log	A trait whose value must be an instance of a specified class.

Continued on next page

Table 47 – continued from previous page

mark_types	
model_id	Gets the model id of this widget.
null_color	A string holding a valid HTML color such as 'blue', '#060482', '#A80'
opacity	A float trait.
preserve_domain	An instance of a Python dict.
row	A numpy array trait type.
row_align	An enum whose value must be in a given sequence.
scales	An instance of a Python dict.
scales_metadata	An instance of a Python dict.
selected	An instance of a Python list.
selected_style	An instance of a Python dict.
stroke	A string holding a valid HTML color such as 'blue', '#060482', '#A80'
tooltip	A trait whose value must be an instance of a specified class.
tooltip_location	An enum whose value must be in a given sequence.
tooltip_style	An instance of a Python dict.
unselected_style	An instance of a Python dict.
visible	A boolean (True, False) trait.
widget_types	
widgets	

bqplot.marks.HeatMap

class bqplot.marks.**HeatMap**(***kwargs*)

HeatMap mark.

Data Attributes

color

color of the data points (2d array).

Type numpy.ndarray or None (default: None)

x

labels for the columns of the *color* array passed. The length of this has to be the number of columns in *color*. This is a scaled attribute.

Type numpy.ndarray or None (default: None)

y

labels for the rows of the *color* array passed. The length of this has to be the number of rows in *color*. This is a scaled attribute.

Type numpy.ndarray or None (default: None)

__init__(***kwargs*)

Public constructor

Methods

__init__(**kwargs)	Public constructor
add_traits(**traits)	Dynamically add trait attributes to the Widget.

Continued on next page

Table 48 – continued from previous page

class_own_trait_events(name)	Get a dict of all event handlers defined on this class, not a parent.
class_own_traits(**metadata)	Get a dict of all the traitlets defined on this class, not a parent.
class_trait_names(**metadata)	Get a list of all the names of this class' traits.
class_traits(**metadata)	Get a dict of all the traits of this class.
close()	Close method.
close_all()	
get_manager_state([drop_defaults, widgets])	Returns the full state for a widget manager for embedding
get_state([key, drop_defaults])	Gets the widget state, or a piece of it.
get_view_spec()	
handle_comm_opened(comm, msg)	Static method, called when a widget is constructed.
has_trait(name)	Returns True if the object has a trait with the specified name.
hold_sync()	Hold syncing any state until the outermost context manager exits
hold_trait_notifications()	Context manager for bundling trait change notifications and cross validation.
notify_change(change)	Called when a property has changed.
observe(handler[, names, type])	Setup a handler to be called when a trait changes.
on_background_click(callback[, remove])	
on_click(callback[, remove])	
on_displayed(callback[, remove])	(Un)Register a widget displayed callback.
on_element_click(callback[, remove])	
on_hover(callback[, remove])	
on_legend_click(callback[, remove])	
on_legend_hover(callback[, remove])	
on_msg(callback[, remove])	(Un)Register a custom msg receive callback.
on_trait_change([handler, name, remove])	DEPRECATED: Setup a handler to be called when a trait changes.
on_widget_constructed(callback)	Registers a callback to be called when a widget is constructed.
open()	Open a comm to the frontend if one isn't already open.
send(content[, buffers])	Sends a custom msg to the widget model in the front-end.
send_state([key])	Sends the widget state, or a piece of it, to the front-end, if it exists.
set_state(sync_data)	Called when a state is received from the front-end.
set_trait(name, value)	Forcibly sets trait attribute, including read-only attributes.
setup_instance(*args, **kwargs)	This is called **before** self.__init__ is called.
trait_events([name])	Get a dict of all the event handlers of this class.
trait_metadata(traitname, key[, default])	Get metadata values for trait by key.
trait_names(**metadata)	Get a list of all the names of this class' traits.
traits(**metadata)	Get a dict of all the traits of this class.
unobserve(handler[, names, type])	Remove a trait change handler.
unobserve_all([name])	Remove trait change handlers of any type for the specified name.

Attributes

apply_clip	A boolean (True, False) trait.
color	A numpy array trait type.
comm	A trait whose value must be an instance of a specified class.
cross_validation_lock	A contextmanager for running a block with our cross validation lock set to True.
display_legend	A boolean (True, False) trait.
enable_hover	A boolean (True, False) trait.
interactions	An instance of a Python dict.
keys	An instance of a Python list.
labels	An instance of a Python list.
log	A trait whose value must be an instance of a specified class.
mark_types	
model_id	Gets the model id of this widget.
null_color	A string holding a valid HTML color such as 'blue', '#060482', '#A80'
preserve_domain	An instance of a Python dict.
scales	An instance of a Python dict.
scales_metadata	An instance of a Python dict.
selected	An instance of a Python list.
selected_style	An instance of a Python dict.
tooltip	A trait whose value must be an instance of a specified class.
tooltip_location	An enum whose value must be in a given sequence.
tooltip_style	An instance of a Python dict.
unselected_style	An instance of a Python dict.
visible	A boolean (True, False) trait.
widget_types	
widgets	
x	A numpy array trait type.
y	A numpy array trait type.

bqplot.marks.Label

class bqplot.marks.**Label**(***kwargs*)

Label mark.

x_offset

horizontal offset in pixels from the stated x location

Type int (default: 0)

y_offset

vertical offset in pixels from the stated y location

Type int (default: 0)

text

text to be displayed

Type string (default: '')

default_size
font size in px, em or ex

Type string (default: '14px')

font_weight
font weight of the caption

Type {'bold', 'normal', 'bolder'}

drag_size
Ratio of the size of the dragged label font size to the default label font size.

Type nonnegative float (default: 1.)

align
alignment of the text with respect to the provided location enable_move: Bool (default: False) Enable the label to be moved by dragging. Refer to restrict_x, restrict_y for more options.

Type {'start', 'middle', 'end'}

restrict_x
Restricts movement of the label to only along the x axis. This is valid only when enable_move is set to True. If both restrict_x and restrict_y are set to True, the label cannot be moved.

Type bool (default: False)

restrict_y
Restricts movement of the label to only along the y axis. This is valid only when enable_move is set to True. If both restrict_x and restrict_y are set to True, the label cannot be moved.

Type bool (default: False)

Data Attributes

x
horizontal position of the labels, in data coordinates or in figure coordinates

Type numpy.ndarray (default: [])

y
vertical position of the labels, in data coordinates or in figure coordinates

Type numpy.ndarray (default: [])

color
label colors

Type numpy.ndarray or None (default: None)

size
label sizes

Type numpy.ndarray or None (default: None)

rotation
label rotations

Type numpy.ndarray or None (default: None)

opacity
label opacities

Type numpy.ndarray or None (default: None)

__init__(***kwargs*)
Public constructor

Methods

__init__(**kwargs)	Public constructor
add_traits(**traits)	Dynamically add trait attributes to the Widget.
class_own_trait_events(name)	Get a dict of all event handlers defined on this class, not a parent.
class_own_traits(**metadata)	Get a dict of all the traitlets defined on this class, not a parent.
class_trait_names(**metadata)	Get a list of all the names of this class' traits.
class_traits(**metadata)	Get a dict of all the traits of this class.
close()	Close method.
close_all()	
get_manager_state([drop_defaults, widgets])	Returns the full state for a widget manager for embedding
get_state([key, drop_defaults])	Gets the widget state, or a piece of it.
get_view_spec()	
handle_comm_opened(comm, msg)	Static method, called when a widget is constructed.
has_trait(name)	Returns True if the object has a trait with the specified name.
hold_sync()	Hold syncing any state until the outermost context manager exits
hold_trait_notifications()	Context manager for bundling trait change notifications and cross validation.
notify_change(change)	Called when a property has changed.
observe(handler[, names, type])	Setup a handler to be called when a trait changes.
on_background_click(callback[, remove])	
on_click(callback[, remove])	
on_displayed(callback[, remove])	(Un)Register a widget displayed callback.
on_drag(callback[, remove])	
on_drag_end(callback[, remove])	
on_drag_start(callback[, remove])	
on_element_click(callback[, remove])	
on_hover(callback[, remove])	
on_legend_click(callback[, remove])	
on_legend_hover(callback[, remove])	
on_msg(callback[, remove])	(Un)Register a custom msg receive callback.
on_trait_change([handler, name, remove])	DEPRECATED: Setup a handler to be called when a trait changes.
on_widget_constructed(callback)	Registers a callback to be called when a widget is constructed.
open()	Open a comm to the frontend if one isn't already open.
send(content[, buffers])	Sends a custom msg to the widget model in the front-end.
send_state([key])	Sends the widget state, or a piece of it, to the front-end, if it exists.
set_state(sync_data)	Called when a state is received from the front-end.

Continued on next page

Table 50 – continued from previous page

set_trait(name, value)	Forcibly sets trait attribute, including read-only attributes.
setup_instance(*args, **kwargs)	This is called **before** self.__init__ is called.
trait_events([name])	Get a dict of all the event handlers of this class.
trait_metadata(traitname, key[, default])	Get metadata values for trait by key.
trait_names(**metadata)	Get a list of all the names of this class' traits.
traits(**metadata)	Get a dict of all the traits of this class.
unobserve(handler[, names, type])	Remove a trait change handler.
unobserve_all([name])	Remove trait change handlers of any type for the specified name.

Attributes

align	An enum whose value must be in a given sequence.
apply_clip	A boolean (True, False) trait.
color	A numpy array trait type.
colors	An instance of a Python list.
comm	A trait whose value must be an instance of a specified class.
cross_validation_lock	A contextmanager for running a block with our cross validation lock set to True.
default_opacities	An instance of a Python list.
default_size	A float trait.
display_legend	A boolean (True, False) trait.
drag_size	A float trait.
enable_delete	A boolean (True, False) trait.
enable_hover	A boolean (True, False) trait.
enable_move	A boolean (True, False) trait.
font_unit	An enum whose value must be in a given sequence.
font_weight	An enum whose value must be in a given sequence.
hovered_point	An int trait.
hovered_style	An instance of a Python dict.
icon	
interactions	An instance of a Python dict.
keys	An instance of a Python list.
labels	An instance of a Python list.
log	A trait whose value must be an instance of a specified class.
mark_types	
model_id	Gets the model id of this widget.
name	
opacity	A numpy array trait type.
preserve_domain	An instance of a Python dict.
restrict_x	A boolean (True, False) trait.
restrict_y	A boolean (True, False) trait.
rotate_angle	A float trait.
rotation	A numpy array trait type.
scales	An instance of a Python dict.
scales_metadata	An instance of a Python dict.
selected	An instance of a Python list.

Continued on next page

Table 51 – continued from previous page

selected_style	An instance of a Python dict.
size	A numpy array trait type.
text	A numpy array trait type.
tooltip	A trait whose value must be an instance of a specified class.
tooltip_location	An enum whose value must be in a given sequence.
tooltip_style	An instance of a Python dict.
unhovered_style	An instance of a Python dict.
unselected_style	An instance of a Python dict.
update_on_move	A boolean (True, False) trait.
visible	A boolean (True, False) trait.
widget_types	
widgets	
x	A numpy array trait type.
x_offset	An int trait.
y	A numpy array trait type.
y_offset	An int trait.

bqplot.marks.OHLC

class bqplot.marks.**OHLC**(***kwargs*)

Open/High/Low/Close marks.

icon
font-awesome icon for that mark

Type string (class-level attribute)

name
user-friendly name of the mark

Type string (class-level attribute)

marker
marker type

Type {'candle', 'bar'}

stroke
stroke color of the marker

Type color (default: None)

stroke_width
stroke width of the marker

Type float (default: 1.0)

colors
fill colors for the markers (up/down)

Type List of colors (default: ['limegreen', 'red'])

opacities
Opacities for the markers of the OHLC mark. Defaults to 1 when the list is too short, or the element of the list is set to None.

Type list of floats (default: [])

format
description of y data being passed supports all permutations of the strings 'ohlc', 'oc', and 'hl'

Type string (default: 'ohlc')

Data Attributes

x
abscissas of the data points (1d array)

Type numpy.ndarray

y
Open/High/Low/Close ordinates of the data points (2d array)

Type numpy.ndarrays

Notes

The fields which can be passed to the default tooltip are: x: the x value associated with the bar/candle open: open value for the bar/candle high: high value for the bar/candle low: low value for the bar/candle close: close value for the bar/candle index: index of the bar/candle being hovered on

__init__(***kwargs*)
Public constructor

Methods

__init__(**kwargs)	Public constructor
add_traits(**traits)	Dynamically add trait attributes to the Widget.
class_own_trait_events(name)	Get a dict of all event handlers defined on this class, not a parent.
class_own_traits(**metadata)	Get a dict of all the traitlets defined on this class, not a parent.
class_trait_names(**metadata)	Get a list of all the names of this class' traits.
class_traits(**metadata)	Get a dict of all the traits of this class.
close()	Close method.
close_all()	
get_manager_state([drop_defaults, widgets])	Returns the full state for a widget manager for embedding
get_state([key, drop_defaults])	Gets the widget state, or a piece of it.
get_view_spec()	
handle_comm_opened(comm, msg)	Static method, called when a widget is constructed.
has_trait(name)	Returns True if the object has a trait with the specified name.
hold_sync()	Hold syncing any state until the outermost context manager exits
hold_trait_notifications()	Context manager for bundling trait change notifications and cross validation.
notify_change(change)	Called when a property has changed.
observe(handler[, names, type])	Setup a handler to be called when a trait changes.
on_background_click(callback[, remove])	
on_click(callback[, remove])	

Continued on next page

Table 52 – continued from previous page

on_displayed(callback[, remove])	(Un)Register a widget displayed callback.
on_element_click(callback[, remove])	
on_hover(callback[, remove])	
on_legend_click(callback[, remove])	
on_legend_hover(callback[, remove])	
on_msg(callback[, remove])	(Un)Register a custom msg receive callback.
on_trait_change([handler, name, remove])	DEPRECATED: Setup a handler to be called when a trait changes.
on_widget_constructed(callback)	Registers a callback to be called when a widget is constructed.
open()	Open a comm to the frontend if one isn't already open.
send(content[, buffers])	Sends a custom msg to the widget model in the front-end.
send_state([key])	Sends the widget state, or a piece of it, to the front-end, if it exists.
set_state(sync_data)	Called when a state is received from the front-end.
set_trait(name, value)	Forcibly sets trait attribute, including read-only attributes.
setup_instance(*args, **kwargs)	This is called **before** self.__init__ is called.
trait_events([name])	Get a dict of all the event handlers of this class.
trait_metadata(traitname, key[, default])	Get metadata values for trait by key.
trait_names(**metadata)	Get a list of all the names of this class' traits.
traits(**metadata)	Get a dict of all the traits of this class.
unobserve(handler[, names, type])	Remove a trait change handler.
unobserve_all([name])	Remove trait change handlers of any type for the specified name.

Attributes

apply_clip	A boolean (True, False) trait.
colors	An instance of a Python list.
comm	A trait whose value must be an instance of a specified class.
cross_validation_lock	A contextmanager for running a block with our cross validation lock set to True.
display_legend	A boolean (True, False) trait.
enable_hover	A boolean (True, False) trait.
format	A trait for unicode strings.
icon	
interactions	An instance of a Python dict.
keys	An instance of a Python list.
labels	An instance of a Python list.
log	A trait whose value must be an instance of a specified class.
mark_types	
marker	An enum whose value must be in a given sequence.
model_id	Gets the model id of this widget.
name	
opacities	An instance of a Python list.

Continued on next page

Table 53 – continued from previous page

preserve_domain	An instance of a Python dict.
scales	An instance of a Python dict.
scales_metadata	An instance of a Python dict.
selected	An instance of a Python list.
selected_style	An instance of a Python dict.
stroke	A string holding a valid HTML color such as 'blue', '#060482', '#A80'
stroke_width	A float trait.
tooltip	A trait whose value must be an instance of a specified class.
tooltip_location	An enum whose value must be in a given sequence.
tooltip_style	An instance of a Python dict.
unselected_style	An instance of a Python dict.
visible	A boolean (True, False) trait.
widget_types	
widgets	
x	A numpy array trait type.
y	A numpy array trait type.

bqplot.marks.Pie

class bqplot.marks.**Pie**(***kwargs*)

Piechart mark.

colors

list of colors for the slices.

Type list of colors (default: CATEGORY10)

stroke

stroke color for the marker

Type color (default: 'white')

opacities

Opacities for the slices of the Pie mark. Defaults to 1 when the list is too short, or the element of the list is set to None.

Type list of floats (default: [])

sort

sort the pie slices by descending sizes

Type bool (default: False)

x

horizontal position of the pie center, in data coordinates or in figure coordinates

Type Float (default: 0.5) or *Date*

y

vertical y position of the pie center, in data coordinates or in figure coordinates

Type Float (default: 0.5)

radius

radius of the pie, in pixels

Type Float

inner_radius
inner radius of the pie, in pixels

Type Float

start_angle
start angle of the pie (from top), in degrees

Type Float (default: 0.0)

end_angle
end angle of the pie (from top), in degrees

Type Float (default: 360.0)

display_labels
label display options

Type {'none', 'inside', 'outside'} (default: 'inside')

display_values
if True show values along with labels

Type bool (default: False)

values_format
format for displaying values

Type string (default: '.2f')

label_color
color of the labels

Type Color or None (default: None)

font_size
label font size in px, em or ex

Type string (default: '14px')

font_weight
label font weight

Type {'bold', 'normal', 'bolder'} (default: 'normal')

Data Attributes

sizes
proportions of the pie slices

Type numpy.ndarray (default: [])

color
color of the data points. Defaults to colors when not provided.

Type numpy.ndarray or None (default: None)

Notes

The fields which can be passed to the default tooltip are: : the x value associated with the bar/candle open: open value for the bar/candle high: high value for the bar/candle low: low value for the bar/candle close: close value for the bar/candle index: index of the bar/candle being hovered on

__init__(***kwargs*)
Public constructor

Methods

__init__(**kwargs)	Public constructor
add_traits(**traits)	Dynamically add trait attributes to the Widget.
class_own_trait_events(name)	Get a dict of all event handlers defined on this class, not a parent.
class_own_traits(**metadata)	Get a dict of all the traitlets defined on this class, not a parent.
class_trait_names(**metadata)	Get a list of all the names of this class' traits.
class_traits(**metadata)	Get a dict of all the traits of this class.
close()	Close method.
close_all()	
get_manager_state([drop_defaults, widgets])	Returns the full state for a widget manager for embedding
get_state([key, drop_defaults])	Gets the widget state, or a piece of it.
get_view_spec()	
handle_comm_opened(comm, msg)	Static method, called when a widget is constructed.
has_trait(name)	Returns True if the object has a trait with the specified name.
hold_sync()	Hold syncing any state until the outermost context manager exits
hold_trait_notifications()	Context manager for bundling trait change notifications and cross validation.
notify_change(change)	Called when a property has changed.
observe(handler[, names, type])	Setup a handler to be called when a trait changes.
on_background_click(callback[, remove])	
on_click(callback[, remove])	
on_displayed(callback[, remove])	(Un)Register a widget displayed callback.
on_element_click(callback[, remove])	
on_hover(callback[, remove])	
on_legend_click(callback[, remove])	
on_legend_hover(callback[, remove])	
on_msg(callback[, remove])	(Un)Register a custom msg receive callback.
on_trait_change([handler, name, remove])	DEPRECATED: Setup a handler to be called when a trait changes.
on_widget_constructed(callback)	Registers a callback to be called when a widget is constructed.
open()	Open a comm to the frontend if one isn't already open.
send(content[, buffers])	Sends a custom msg to the widget model in the front-end.
send_state([key])	Sends the widget state, or a piece of it, to the front-end, if it exists.
set_state(sync_data)	Called when a state is received from the front-end.
set_trait(name, value)	Forcibly sets trait attribute, including read-only attributes.
setup_instance(*args, **kwargs)	This is called **before** self.__init__ is called.

Continued on next page

Table 54 – continued from previous page

trait_events([name])	Get a dict of all the event handlers of this class.
trait_metadata(traitname, key[, default])	Get metadata values for trait by key.
trait_names(**metadata)	Get a list of all the names of this class' traits.
traits(**metadata)	Get a dict of all the traits of this class.
unobserve(handler[, names, type])	Remove a trait change handler.
unobserve_all([name])	Remove trait change handlers of any type for the specified name.

Attributes

apply_clip	A boolean (True, False) trait.
color	A numpy array trait type.
colors	An instance of a Python list.
comm	A trait whose value must be an instance of a specified class.
cross_validation_lock	A contextmanager for running a block with our cross validation lock set to True.
display_labels	An enum whose value must be in a given sequence.
display_legend	A boolean (True, False) trait.
display_values	A boolean (True, False) trait.
enable_hover	A boolean (True, False) trait.
end_angle	A float trait.
font_size	A trait for unicode strings.
font_weight	An enum whose value must be in a given sequence.
icon	
inner_radius	A float trait.
interactions	An instance of a Python dict.
keys	An instance of a Python list.
label_color	A string holding a valid HTML color such as 'blue', '#060482', '#A80'
labels	An instance of a Python list.
log	A trait whose value must be an instance of a specified class.
mark_types	
model_id	Gets the model id of this widget.
name	
opacities	An instance of a Python list.
preserve_domain	An instance of a Python dict.
radius	A float trait.
scales	An instance of a Python dict.
scales_metadata	An instance of a Python dict.
selected	An instance of a Python list.
selected_style	An instance of a Python dict.
sizes	A numpy array trait type.
sort	A boolean (True, False) trait.
start_angle	A float trait.
stroke	A string holding a valid HTML color such as 'blue', '#060482', '#A80'
tooltip	A trait whose value must be an instance of a specified class.

Continued on next page

Table 55 – continued from previous page

tooltip_location	An enum whose value must be in a given sequence.
tooltip_style	An instance of a Python dict.
unselected_style	An instance of a Python dict.
values_format	A trait for unicode strings.
visible	A boolean (True, False) trait.
widget_types	
widgets	
x	A trait type representing a Union type.
y	A trait type representing a Union type.

bqplot.marks.Map

class bqplot.marks.**Map**(***kwargs*)

Map mark.

colors

default colors for items of the map when no color data is passed. The dictionary should be indexed by the id of the element and have the corresponding colors as values. The key *default_color* controls the items for which no color is specified.

Type Dict (default: {})

selected_styles

Type Dict (default: {'selected_fill': 'Red',

'selected_stroke'

Dictionary containing the styles for selected subunits

Type None, 'selected_stroke_width': 2.0})

hovered_styles

Type Dict (default: {'hovered_fill': 'Orange',

'hovered_stroke'

Dictionary containing the styles for hovered subunits

Type None, 'hovered_stroke_width': 2.0})

selected

list containing the selected countries in the map

Type List (default: [])

hover_highlight

boolean to control if the map should be aware of which country is being hovered on.

Type bool (default: True)

map_data

a topojson-formatted dictionary with the objects to map under the key 'subunits'.

Type dict (default: topo_load("map_data/WorldMap.json"))

Data Attributes

color

dictionary containing the data associated with every country for the color scale

Type Dict or None (default: None)

__init__(***kwargs*)
Public constructor

Methods

__init__(**kwargs)	Public constructor
add_traits(**traits)	Dynamically add trait attributes to the Widget.
class_own_trait_events(name)	Get a dict of all event handlers defined on this class, not a parent.
class_own_traits(**metadata)	Get a dict of all the traitlets defined on this class, not a parent.
class_trait_names(**metadata)	Get a list of all the names of this class' traits.
class_traits(**metadata)	Get a dict of all the traits of this class.
close()	Close method.
close_all()	
get_manager_state([drop_defaults, widgets])	Returns the full state for a widget manager for embedding
get_state([key, drop_defaults])	Gets the widget state, or a piece of it.
get_view_spec()	
handle_comm_opened(comm, msg)	Static method, called when a widget is constructed.
has_trait(name)	Returns True if the object has a trait with the specified name.
hold_sync()	Hold syncing any state until the outermost context manager exits
hold_trait_notifications()	Context manager for bundling trait change notifications and cross validation.
notify_change(change)	Called when a property has changed.
observe(handler[, names, type])	Setup a handler to be called when a trait changes.
on_background_click(callback[, remove])	
on_click(callback[, remove])	
on_displayed(callback[, remove])	(Un)Register a widget displayed callback.
on_element_click(callback[, remove])	
on_hover(callback[, remove])	
on_legend_click(callback[, remove])	
on_legend_hover(callback[, remove])	
on_msg(callback[, remove])	(Un)Register a custom msg receive callback.
on_trait_change([handler, name, remove])	DEPRECATED: Setup a handler to be called when a trait changes.
on_widget_constructed(callback)	Registers a callback to be called when a widget is constructed.
open()	Open a comm to the frontend if one isn't already open.
send(content[, buffers])	Sends a custom msg to the widget model in the front-end.
send_state([key])	Sends the widget state, or a piece of it, to the front-end, if it exists.
set_state(sync_data)	Called when a state is received from the front-end.
set_trait(name, value)	Forcibly sets trait attribute, including read-only attributes.
setup_instance(*args, **kwargs)	This is called **before** self.__init__ is called.

Continued on next page

Table 56 – continued from previous page

trait_events([name])	Get a dict of all the event handlers of this class.
trait_metadata(traitname, key[, default])	Get metadata values for trait by key.
trait_names(**metadata)	Get a list of all the names of this class' traits.
traits(**metadata)	Get a dict of all the traits of this class.
unobserve(handler[, names, type])	Remove a trait change handler.
unobserve_all([name])	Remove trait change handlers of any type for the specified name.

Attributes

apply_clip	A boolean (True, False) trait.
color	An instance of a Python dict.
colors	An instance of a Python dict.
comm	A trait whose value must be an instance of a specified class.
cross_validation_lock	A contextmanager for running a block with our cross validation lock set to True.
display_legend	A boolean (True, False) trait.
enable_hover	A boolean (True, False) trait.
hover_highlight	A boolean (True, False) trait.
hovered_styles	An instance of a Python dict.
icon	
interactions	An instance of a Python dict.
keys	An instance of a Python list.
labels	An instance of a Python list.
log	A trait whose value must be an instance of a specified class.
map_data	An instance of a Python dict.
mark_types	
model_id	Gets the model id of this widget.
name	
preserve_domain	An instance of a Python dict.
scales	An instance of a Python dict.
scales_metadata	An instance of a Python dict.
selected	An instance of a Python list.
selected_style	An instance of a Python dict.
selected_styles	An instance of a Python dict.
stroke_color	A string holding a valid HTML color such as 'blue', '#060482', '#A80'
tooltip	A trait whose value must be an instance of a specified class.
tooltip_location	An enum whose value must be in a given sequence.
tooltip_style	An instance of a Python dict.
unselected_style	An instance of a Python dict.
visible	A boolean (True, False) trait.
widget_types	
widgets	

3.1.4 Axes

Axis(**kwargs)	A line axis.
ColorAxis(**kwargs)	A colorbar axis.

bqplot.axes.Axis

class bqplot.axes.**Axis**(***kwargs*)

A line axis.

A line axis is the visual representation of a numerical or date scale.

icon

The font-awesome icon name for this object.

Type string (class-level attribute)

axis_types

A registry of existing axis types.

Type dict (class-level attribute)

orientation

The orientation of the axis, either vertical or horizontal

Type {'horizontal', 'vertical'}

side

The side of the axis, either bottom, top, left or right.

Type {'bottom', 'top', 'left', 'right'} or None (default: None)

label

The axis label

Type string (default: '')

tick_format

The tick format for the axis, for dates use d3 string formatting.

Type string or None (default: '')

scale

The scale represented by the axis

Type *Scale*

num_ticks

If tick_values is None, number of ticks

Type int or None (default: None)

tick_values

Tick values for the axis

Type numpy.ndarray or None (default: None)

offset

Contains a scale and a value {'scale': scale or None, 'value': value of the offset} If offset['scale'] is None, the corresponding figure scale is used instead.

Type dict (default: {})

label_location
The location of the label along the axis, one of 'start', 'end' or 'middle'

Type {'middle', 'start', 'end'}

label_color
The color of the axis label

Type Color or None (default: None)

grid_lines
The display of the grid lines

Type {'none', 'solid', 'dashed'}

grid_color
The color of the grid lines

Type Color or None (default: None)

color
The color of the line

Type Color or None (default: None)

label_offset
Label displacement from the axis line. Units allowed are 'em', 'px' and 'ex'. Positive values are away from the figure and negative values are towards the figure with resepect to the axis line.

Type string or None (default: None)

visible
A visibility toggle for the axis

Type bool (default: True)

tick_style
Dictionary containing the CSS-style of the text for the ticks. For example: font-size of the text can be changed by passing *{'font-size': 14}*

Type Dict (default: {})

tick_rotate
Degrees to rotate tick labels by.

Type int (default: 0)

__init__(***kwargs*)
Public constructor

Methods

__init__(**kwargs)	Public constructor
add_traits(**traits)	Dynamically add trait attributes to the Widget.
class_own_trait_events(name)	Get a dict of all event handlers defined on this class, not a parent.
class_own_traits(**metadata)	Get a dict of all the traitlets defined on this class, not a parent.
class_trait_names(**metadata)	Get a list of all the names of this class' traits.
class_traits(**metadata)	Get a dict of all the traits of this class.

Continued on next page

Table 59 – continued from previous page

close()	Close method.
close_all()	
get_manager_state([drop_defaults, widgets])	Returns the full state for a widget manager for embedding
get_state([key, drop_defaults])	Gets the widget state, or a piece of it.
get_view_spec()	
handle_comm_opened(comm, msg)	Static method, called when a widget is constructed.
has_trait(name)	Returns True if the object has a trait with the specified name.
hold_sync()	Hold syncing any state until the outermost context manager exits
hold_trait_notifications()	Context manager for bundling trait change notifications and cross validation.
notify_change(change)	Called when a property has changed.
observe(handler[, names, type])	Setup a handler to be called when a trait changes.
on_displayed(callback[, remove])	(Un)Register a widget displayed callback.
on_msg(callback[, remove])	(Un)Register a custom msg receive callback.
on_trait_change([handler, name, remove])	DEPRECATED: Setup a handler to be called when a trait changes.
on_widget_constructed(callback)	Registers a callback to be called when a widget is constructed.
open()	Open a comm to the frontend if one isn't already open.
send(content[, buffers])	Sends a custom msg to the widget model in the front-end.
send_state([key])	Sends the widget state, or a piece of it, to the front-end, if it exists.
set_state(sync_data)	Called when a state is received from the front-end.
set_trait(name, value)	Forcibly sets trait attribute, including read-only attributes.
setup_instance(*args, **kwargs)	This is called **before** self.__init__ is called.
trait_events([name])	Get a dict of all the event handlers of this class.
trait_metadata(traitname, key[, default])	Get metadata values for trait by key.
trait_names(**metadata)	Get a list of all the names of this class' traits.
traits(**metadata)	Get a dict of all the traits of this class.
unobserve(handler[, names, type])	Remove a trait change handler.
unobserve_all([name])	Remove trait change handlers of any type for the specified name.

Attributes

axis_types	
color	A string holding a valid HTML color such as 'blue', '#060482', '#A80'
comm	A trait whose value must be an instance of a specified class.
cross_validation_lock	A contextmanager for running a block with our cross validation lock set to True.
grid_color	A string holding a valid HTML color such as 'blue', '#060482', '#A80'

Continued on next page

Table 60 – continued from previous page

grid_lines	An enum whose value must be in a given sequence.
icon	
keys	An instance of a Python list.
label	A trait for unicode strings.
label_color	A string holding a valid HTML color such as 'blue', '#060482', '#A80'
label_location	An enum whose value must be in a given sequence.
label_offset	A trait for unicode strings.
log	A trait whose value must be an instance of a specified class.
model_id	Gets the model id of this widget.
num_ticks	An int trait.
offset	An instance of a Python dict.
orientation	An enum whose value must be in a given sequence.
scale	A trait whose value must be an instance of a specified class.
side	An enum whose value must be in a given sequence.
tick_format	A trait for unicode strings.
tick_rotate	An int trait.
tick_style	An instance of a Python dict.
tick_values	A numpy array trait type.
visible	A boolean (True, False) trait.
widget_types	
widgets	

bqplot.axes.ColorAxis

class bqplot.axes.**ColorAxis**(***kwargs*)

A colorbar axis.

A color axis is the visual representation of a color scale.

scale

The scale represented by the axis

Type *ColorScale*

__init__(***kwargs*)

Public constructor

Methods

__init__(**kwargs)	Public constructor
add_traits(**traits)	Dynamically add trait attributes to the Widget.
class_own_trait_events(name)	Get a dict of all event handlers defined on this class, not a parent.
class_own_traits(**metadata)	Get a dict of all the traitlets defined on this class, not a parent.
class_trait_names(**metadata)	Get a list of all the names of this class' traits.
class_traits(**metadata)	Get a dict of all the traits of this class.
close()	Close method.

Continued on next page

Table 61 – continued from previous page

close_all()	
get_manager_state([drop_defaults, widgets])	Returns the full state for a widget manager for embedding
get_state([key, drop_defaults])	Gets the widget state, or a piece of it.
get_view_spec()	
handle_comm_opened(comm, msg)	Static method, called when a widget is constructed.
has_trait(name)	Returns True if the object has a trait with the specified name.
hold_sync()	Hold syncing any state until the outermost context manager exits
hold_trait_notifications()	Context manager for bundling trait change notifications and cross validation.
notify_change(change)	Called when a property has changed.
observe(handler[, names, type])	Setup a handler to be called when a trait changes.
on_displayed(callback[, remove])	(Un)Register a widget displayed callback.
on_msg(callback[, remove])	(Un)Register a custom msg receive callback.
on_trait_change([handler, name, remove])	DEPRECATED: Setup a handler to be called when a trait changes.
on_widget_constructed(callback)	Registers a callback to be called when a widget is constructed.
open()	Open a comm to the frontend if one isn't already open.
send(content[, buffers])	Sends a custom msg to the widget model in the front-end.
send_state([key])	Sends the widget state, or a piece of it, to the front-end, if it exists.
set_state(sync_data)	Called when a state is received from the front-end.
set_trait(name, value)	Forcibly sets trait attribute, including read-only attributes.
setup_instance(*args, **kwargs)	This is called **before** self.__init__ is called.
trait_events([name])	Get a dict of all the event handlers of this class.
trait_metadata(traitname, key[, default])	Get metadata values for trait by key.
trait_names(**metadata)	Get a list of all the names of this class' traits.
traits(**metadata)	Get a dict of all the traits of this class.
unobserve(handler[, names, type])	Remove a trait change handler.
unobserve_all([name])	Remove trait change handlers of any type for the specified name.

Attributes

axis_types	
color	A string holding a valid HTML color such as 'blue', '#060482', '#A80'
comm	A trait whose value must be an instance of a specified class.
cross_validation_lock	A contextmanager for running a block with our cross validation lock set to True.
grid_color	A string holding a valid HTML color such as 'blue', '#060482', '#A80'
grid_lines	An enum whose value must be in a given sequence.

Continued on next page

Table 62 – continued from previous page

icon	
keys	An instance of a Python list.
label	A trait for unicode strings.
label_color	A string holding a valid HTML color such as 'blue', '#060482', '#A80'
label_location	An enum whose value must be in a given sequence.
label_offset	A trait for unicode strings.
log	A trait whose value must be an instance of a specified class.
model_id	Gets the model id of this widget.
num_ticks	An int trait.
offset	An instance of a Python dict.
orientation	An enum whose value must be in a given sequence.
scale	A trait whose value must be an instance of a specified class.
side	An enum whose value must be in a given sequence.
tick_format	A trait for unicode strings.
tick_rotate	An int trait.
tick_style	An instance of a Python dict.
tick_values	A numpy array trait type.
visible	A boolean (True, False) trait.
widget_types	
widgets	

3.1.5 Market Map

MarketMap(**kwargs)	Waffle wrapped map.
SquareMarketMap(**kwargs)	

bqplot.market_map.MarketMap

class bqplot.market_map.**MarketMap**(***kwargs*)

Waffle wrapped map.

names

The elements can also be objects convertible to string primary key for the map data. A rectangle is created for each unique entry in this array

Type numpy.ndarray of strings (default: [])

groups

attribute on which the groupby is run. If this is an empty arrray, then there is no group by for the map.

Type numpy.ndarray (default: [])

display_text

data to be displayed on each rectangle of the map.If this is empty it defaults to the names attribute.

Type numpy.ndarray or None(default: None)

ref_data

Additional data associated with each element of the map. The data in this data frame can be displayed as a tooltip.

Type pandas.DataDrame or None (default: None)

color
Data to represent the color for each of the cells. If the value of the data is NaN for a cell, then the color of the cell is the color of the group it belongs to in absence of data for color

Type numpy.ndarray (default: [])

scales
If the map has data being passed as color, then a corresponding color scale is required

Type Dictionary of scales holding a scale for each data attribute

axes
Ability to add an axis for the scales which are used to scale data represented in the map

Type List of axes

on_hover
This event is received when the mouse is hovering over a cell. Returns the data of the cell and the ref_data associated with the cell.

Type custom event

tooltip_widget
Widget to be displayed as the tooltip. This can be combined with the on_hover event to display the chart corresponding to the cell being hovered on.

Type Instance of a widget

tooltip_fields
names of the fields from the ref_data dataframe which should be displayed in the tooltip.

Type list

tooltip_formats
formats for each of the fields for the tooltip data. Order should match the order of the tooltip_fields

Type list

show_groups
attribute to determine if the groups should be displayed. If set to True, the finer elements are blurred

Type bool

Map Drawing Attributes

cols
Suggestion for no of columns in the map.If not specified, value is inferred from the no of rows and no of cells

Type int

rows
No of rows in the map.If not specified, value is inferred from the no of cells and no of columns. If both rows and columns are not specified, then a square is constructed basing on the no of cells. The above two attributes are suggestions which are respected unless they are not feasible. One required condition is that, the number of columns is odd when row_groups is greater than 1.

Type int

row_groups
No of groups the rows should be divided into. This can be used to draw more square cells for each of the groups

Type int

Layout Attributes

map_margin
Dictionary containing the top, bottom, left and right margins. The user is responsible for making sure that the width and height are greater than the sum of the margins.

Type dict (default: {top=50, bottom=50, left=50, right=50})

min_aspect_ratio
minimum width / height ratio of the figure

Type float

max_aspect_ratio
maximum width / height ratio of the figure

Type float

Display Attributes

colors: list of colors Colors for each of the groups which are cycled over to cover all the groups

title: string Title of the Market Map

title_style: dict CSS style for the title of the Market Map

stroke: color Stroke of each of the cells of the market map

group_stroke: color Stroke of the border for the group of cells corresponding to a group

selected_stroke: color stroke for the selected cells

hovered_stroke: color stroke for the cell being hovered on

font_style: dict CSS style for the text of each cell

Other Attributes

enable_select: bool boolean to control the ability to select the cells of the map by clicking

enable_hover: bool boolean to control if the map should be aware of which cell is being hovered on. If it is set to False, tooltip will not be displayed

---

Note: The aspect ratios stand for width / height ratios.

- If the available space is within bounds in terms of min and max aspect ratio, we use the entire available space.
- If the available space is too oblong horizontally, we use the client height and the width that corresponds max_aspect_ratio (maximize width under the constraints).
- If the available space is too oblong vertically, we use the client width and the height that corresponds to min_aspect_ratio (maximize height under the constraint). This corresponds to maximizing the area under the constraints.

Default min and max aspect ratio are both equal to 16 / 9.

__init__ (***kwargs*)
Public constructor

Methods

__init__(**kwargs)	Public constructor
add_class(className)	Adds a class to the top level element of the widget.
add_traits(**traits)	Dynamically add trait attributes to the Widget.
class_own_trait_events(name)	Get a dict of all event handlers defined on this class, not a parent.
class_own_traits(**metadata)	Get a dict of all the traitlets defined on this class, not a parent.
class_trait_names(**metadata)	Get a list of all the names of this class' traits.
class_traits(**metadata)	Get a dict of all the traits of this class.
close()	Close method.
close_all()	
get_manager_state([drop_defaults, widgets])	Returns the full state for a widget manager for embedding
get_state([key, drop_defaults])	Gets the widget state, or a piece of it.
get_view_spec()	
handle_comm_opened(comm, msg)	Static method, called when a widget is constructed.
has_trait(name)	Returns True if the object has a trait with the specified name.
hold_sync()	Hold syncing any state until the outermost context manager exits
hold_trait_notifications()	Context manager for bundling trait change notifications and cross validation.
notify_change(change)	Called when a property has changed.
observe(handler[, names, type])	Setup a handler to be called when a trait changes.
on_displayed(callback[, remove])	(Un)Register a widget displayed callback.
on_hover(callback[, remove])	
on_msg(callback[, remove])	(Un)Register a custom msg receive callback.
on_trait_change([handler, name, remove])	DEPRECATED: Setup a handler to be called when a trait changes.
on_widget_constructed(callback)	Registers a callback to be called when a widget is constructed.
open()	Open a comm to the frontend if one isn't already open.
remove_class(className)	Removes a class from the top level element of the widget.
send(content[, buffers])	Sends a custom msg to the widget model in the front-end.
send_state([key])	Sends the widget state, or a piece of it, to the front-end, if it exists.
set_state(sync_data)	Called when a state is received from the front-end.
set_trait(name, value)	Forcibly sets trait attribute, including read-only attributes.
setup_instance(*args, **kwargs)	This is called **before** self.__init__ is called.
trait_events([name])	Get a dict of all the event handlers of this class.
trait_metadata(traitname, key[, default])	Get metadata values for trait by key.
trait_names(**metadata)	Get a list of all the names of this class' traits.
traits(**metadata)	Get a dict of all the traits of this class.
unobserve(handler[, names, type])	Remove a trait change handler.
unobserve_all([name])	Remove trait change handlers of any type for the specified name.

Attributes

axes	An instance of a Python list.
color	A numpy array trait type.
colors	An instance of a Python list.
cols	An int trait.
comm	A trait whose value must be an instance of a specified class.
cross_validation_lock	A contextmanager for running a block with our cross validation lock set to True.
display_text	A numpy array trait type.
enable_hover	A boolean (True, False) trait.
enable_select	A boolean (True, False) trait.
font_style	An instance of a Python dict.
group_stroke	A string holding a valid HTML color such as 'blue', '#060482', '#A80'
groups	A numpy array trait type.
hovered_stroke	A string holding a valid HTML color such as 'blue', '#060482', '#A80'
keys	An instance of a Python list.
layout	
log	A trait whose value must be an instance of a specified class.
map_margin	An instance of a Python dict.
max_aspect_ratio	A float trait.
min_aspect_ratio	A float trait.
model_id	Gets the model id of this widget.
names	A numpy array trait type.
ref_data	A pandas dataframe trait type.
row_groups	An int trait.
rows	An int trait.
scales	An instance of a Python dict.
selected	An instance of a Python list.
selected_stroke	A string holding a valid HTML color such as 'blue', '#060482', '#A80'
show_groups	A boolean (True, False) trait.
stroke	A string holding a valid HTML color such as 'blue', '#060482', '#A80'
title	A trait for unicode strings.
title_style	An instance of a Python dict.
tooltip_fields	An instance of a Python list.
tooltip_formats	An instance of a Python list.
tooltip_widget	A trait whose value must be an instance of a specified class.
widget_types	
widgets	

bqplot.market_map.SquareMarketMap

class bqplot.market_map.**SquareMarketMap**(***kwargs*)

__init__(***kwargs*)
Public constructor

Methods

__init__(**kwargs)	Public constructor
add_class(className)	Adds a class to the top level element of the widget.
add_traits(**traits)	Dynamically add trait attributes to the Widget.
class_own_trait_events(name)	Get a dict of all event handlers defined on this class, not a parent.
class_own_traits(**metadata)	Get a dict of all the traitlets defined on this class, not a parent.
class_trait_names(**metadata)	Get a list of all the names of this class' traits.
class_traits(**metadata)	Get a dict of all the traits of this class.
close()	Close method.
close_all()	
get_manager_state([drop_defaults, widgets])	Returns the full state for a widget manager for embedding
get_state([key, drop_defaults])	Gets the widget state, or a piece of it.
get_view_spec()	
handle_comm_opened(comm, msg)	Static method, called when a widget is constructed.
has_trait(name)	Returns True if the object has a trait with the specified name.
hold_sync()	Hold syncing any state until the outermost context manager exits
hold_trait_notifications()	Context manager for bundling trait change notifications and cross validation.
notify_change(change)	Called when a property has changed.
observe(handler[, names, type])	Setup a handler to be called when a trait changes.
on_displayed(callback[, remove])	(Un)Register a widget displayed callback.
on_hover(callback[, remove])	
on_msg(callback[, remove])	(Un)Register a custom msg receive callback.
on_trait_change([handler, name, remove])	DEPRECATED: Setup a handler to be called when a trait changes.
on_widget_constructed(callback)	Registers a callback to be called when a widget is constructed.
open()	Open a comm to the frontend if one isn't already open.
remove_class(className)	Removes a class from the top level element of the widget.
send(content[, buffers])	Sends a custom msg to the widget model in the front-end.
send_state([key])	Sends the widget state, or a piece of it, to the front-end, if it exists.
set_state(sync_data)	Called when a state is received from the front-end.
set_trait(name, value)	Forcibly sets trait attribute, including read-only attributes.
setup_instance(*args, **kwargs)	This is called **before** self.__init__ is called.
trait_events([name])	Get a dict of all the event handlers of this class.
trait_metadata(traitname, key[, default])	Get metadata values for trait by key.

Continued on next page

Table 66 – continued from previous page

trait_names(**metadata)	Get a list of all the names of this class' traits.
traits(**metadata)	Get a dict of all the traits of this class.
unobserve(handler[, names, type])	Remove a trait change handler.
unobserve_all([name])	Remove trait change handlers of any type for the specified name.

Attributes

axes	An instance of a Python list.
color	A numpy array trait type.
colors	An instance of a Python list.
cols	An int trait.
comm	A trait whose value must be an instance of a specified class.
cross_validation_lock	A contextmanager for running a block with our cross validation lock set to True.
data	An instance of a Python dict.
display_text	A numpy array trait type.
enable_hover	A boolean (True, False) trait.
enable_select	A boolean (True, False) trait.
font_style	An instance of a Python dict.
group_stroke	A string holding a valid HTML color such as 'blue', '#060482', '#A80'
groups	A numpy array trait type.
hovered_stroke	A string holding a valid HTML color such as 'blue', '#060482', '#A80'
keys	An instance of a Python list.
layout	
log	A trait whose value must be an instance of a specified class.
map_margin	An instance of a Python dict.
margin	An instance of a Python dict.
max_aspect_ratio	A float trait.
min_aspect_ratio	A float trait.
mode	An enum whose value must be in a given sequence.
model_id	Gets the model id of this widget.
names	A numpy array trait type.
ref_data	A pandas dataframe trait type.
row_groups	An int trait.
rows	An int trait.
scales	An instance of a Python dict.
selected	An instance of a Python list.
selected_stroke	A string holding a valid HTML color such as 'blue', '#060482', '#A80'
show_groups	A boolean (True, False) trait.
stroke	A string holding a valid HTML color such as 'blue', '#060482', '#A80'
title	A trait for unicode strings.
title_style	An instance of a Python dict.
tooltip_fields	An instance of a Python list.

Continued on next page

Table 67 – continued from previous page

tooltip_formats	An instance of a Python list.
tooltip_widget	A trait whose value must be an instance of a specified class.
widget_types	
widgets	

3.1.6 Interacts

BrushIntervalSelector(**kwargs)	Brush interval selector interaction.
BrushSelector(**kwargs)	Brush interval selector interaction.
HandDraw(**kwargs)	A hand-draw interaction.
IndexSelector(**kwargs)	Index selector interaction.
FastIntervalSelector(**kwargs)	Fast interval selector interaction.
MultiSelector(**kwargs)	Multi selector interaction.
OneDSelector(**kwargs)	One-dimensional selector interaction
Interaction(**kwargs)	The base interaction class.
PanZoom(**kwargs)	An interaction to pan and zoom wrt scales.
Selector(**kwargs)	Selector interaction.
TwoDSelector(**kwargs)	Two-dimensional selector interaction.

bqplot.interacts.BrushIntervalSelector

class bqplot.interacts.**BrushIntervalSelector**(***kwargs*)

Brush interval selector interaction.

This 1-D selector interaction enables the user to select an interval using the brushing action of the mouse. A mouse-down marks the start of the interval. The drag after the mouse down in the x-direction selects the extent and a mouse-up signifies the end of the interval.

Once an interval is drawn, the selector can be moved to a new interval by dragging the selector to the new interval.

A double click at the same point without moving the mouse in the x-direction will result in the entire interval being selected.

selected

Two element array containing the start and end of the interval selected in terms of the scale of the selector. This attribute changes while the selection is being made with the BrushIntervalSelector.

Type numpy.ndarray

brushing

Boolean attribute to indicate if the selector is being dragged. It is True when the selector is being moved and False when it is not. This attribute can be used to trigger computationally intensive code which should be run only on the interval selection being completed as opposed to code which should be run whenever selected is changing.

Type bool

orientation

The orientation of the interval, either vertical or horizontal

Type {'horizontal', 'vertical'}

color

Color of the rectangle representing the brush selector.

Type Color or None (default: None)

__init__(***kwargs*)
Public constructor

Methods

__init__(**kwargs)	Public constructor
add_traits(**traits)	Dynamically add trait attributes to the Widget.
class_own_trait_events(name)	Get a dict of all event handlers defined on this class, not a parent.
class_own_traits(**metadata)	Get a dict of all the traitlets defined on this class, not a parent.
class_trait_names(**metadata)	Get a list of all the names of this class' traits.
class_traits(**metadata)	Get a dict of all the traits of this class.
close()	Close method.
close_all()	
get_manager_state([drop_defaults, widgets])	Returns the full state for a widget manager for embedding
get_state([key, drop_defaults])	Gets the widget state, or a piece of it.
get_view_spec()	
handle_comm_opened(comm, msg)	Static method, called when a widget is constructed.
has_trait(name)	Returns True if the object has a trait with the specified name.
hold_sync()	Hold syncing any state until the outermost context manager exits
hold_trait_notifications()	Context manager for bundling trait change notifications and cross validation.
notify_change(change)	Called when a property has changed.
observe(handler[, names, type])	Setup a handler to be called when a trait changes.
on_displayed(callback[, remove])	(Un)Register a widget displayed callback.
on_msg(callback[, remove])	(Un)Register a custom msg receive callback.
on_trait_change([handler, name, remove])	DEPRECATED: Setup a handler to be called when a trait changes.
on_widget_constructed(callback)	Registers a callback to be called when a widget is constructed.
open()	Open a comm to the frontend if one isn't already open.
reset()	
send(content[, buffers])	Sends a custom msg to the widget model in the front-end.
send_state([key])	Sends the widget state, or a piece of it, to the front-end, if it exists.
set_state(sync_data)	Called when a state is received from the front-end.
set_trait(name, value)	Forcibly sets trait attribute, including read-only attributes.
setup_instance(*args, **kwargs)	This is called **before** self.__init__ is called.
trait_events([name])	Get a dict of all the event handlers of this class.
trait_metadata(traitname, key[, default])	Get metadata values for trait by key.
trait_names(**metadata)	Get a list of all the names of this class' traits.
traits(**metadata)	Get a dict of all the traits of this class.

Continued on next page

Table 69 – continued from previous page

unobserve(handler[, names, type])	Remove a trait change handler.
unobserve_all([name])	Remove trait change handlers of any type for the specified name.

Attributes

brushing	A boolean (True, False) trait.
color	A string holding a valid HTML color such as 'blue', '#060482', '#A80'
comm	A trait whose value must be an instance of a specified class.
cross_validation_lock	A contextmanager for running a block with our cross validation lock set to True.
keys	An instance of a Python list.
log	A trait whose value must be an instance of a specified class.
marks	An instance of a Python list.
model_id	Gets the model id of this widget.
orientation	An enum whose value must be in a given sequence.
scale	A trait whose value must be an instance of a specified class.
selected	A numpy array trait type.
types	
widget_types	
widgets	

bqplot.interacts.BrushSelector

class bqplot.interacts.**BrushSelector**(***kwargs*)

Brush interval selector interaction.

This 2-D selector interaction enables the user to select a rectangular region using the brushing action of the mouse. A mouse-down marks the starting point of the interval. The drag after the mouse down selects the rectangle of interest and a mouse-up signifies the end point of the interval.

Once an interval is drawn, the selector can be moved to a new interval by dragging the selector to the new interval.

A double click at the same point without moving the mouse will result in the entire interval being selected.

selected_x

Two element array containing the start and end of the interval selected in terms of the x_scale of the selector. This attribute changes while the selection is being made with the BrushSelector.

Type numpy.ndarray

selected_y

Two element array containing the start and end of the interval selected in terms of the y_scale of the selector. This attribute changes while the selection is being made with the BrushSelector.

Type numpy.ndarray

selected

Readonly 2x2 array containing the coordinates [[selected_x[0], selected_y[0]],

[selected_x[1], selected_y[1]]]

Type list

brushing
boolean attribute to indicate if the selector is being dragged. It is True when the selector is being moved and False when it is not. This attribute can be used to trigger computationally intensive code which should be run only on the interval selection being completed as opposed to code which should be run whenever selected is changing.

Type bool (default: False)

color
Color of the rectangle representing the brush selector.

Type Color or None (default: None)

__init__(***kwargs*)
Public constructor

Methods

__init__(**kwargs)	Public constructor
add_traits(**traits)	Dynamically add trait attributes to the Widget.
class_own_trait_events(name)	Get a dict of all event handlers defined on this class, not a parent.
class_own_traits(**metadata)	Get a dict of all the traitlets defined on this class, not a parent.
class_trait_names(**metadata)	Get a list of all the names of this class' traits.
class_traits(**metadata)	Get a dict of all the traits of this class.
close()	Close method.
close_all()	
get_manager_state([drop_defaults, widgets])	Returns the full state for a widget manager for embedding
get_state([key, drop_defaults])	Gets the widget state, or a piece of it.
get_view_spec()	
handle_comm_opened(comm, msg)	Static method, called when a widget is constructed.
has_trait(name)	Returns True if the object has a trait with the specified name.
hold_sync()	Hold syncing any state until the outermost context manager exits
hold_trait_notifications()	Context manager for bundling trait change notifications and cross validation.
notify_change(change)	Called when a property has changed.
observe(handler[, names, type])	Setup a handler to be called when a trait changes.
on_displayed(callback[, remove])	(Un)Register a widget displayed callback.
on_msg(callback[, remove])	(Un)Register a custom msg receive callback.
on_trait_change([handler, name, remove])	DEPRECATED: Setup a handler to be called when a trait changes.
on_widget_constructed(callback)	Registers a callback to be called when a widget is constructed.
open()	Open a comm to the frontend if one isn't already open.

Continued on next page

Table 71 – continued from previous page

reset()	
send(content[, buffers])	Sends a custom msg to the widget model in the front-end.
send_state([key])	Sends the widget state, or a piece of it, to the front-end, if it exists.
set_state(sync_data)	Called when a state is received from the front-end.
set_trait(name, value)	Forcibly sets trait attribute, including read-only attributes.
setup_instance(*args, **kwargs)	This is called **before** self.__init__ is called.
trait_events([name])	Get a dict of all the event handlers of this class.
trait_metadata(traitname, key[, default])	Get metadata values for trait by key.
trait_names(**metadata)	Get a list of all the names of this class' traits.
traits(**metadata)	Get a dict of all the traits of this class.
unobserve(handler[, names, type])	Remove a trait change handler.
unobserve_all([name])	Remove trait change handlers of any type for the specified name.

Attributes

brushing	A boolean (True, False) trait.
clear	A boolean (True, False) trait.
color	A string holding a valid HTML color such as 'blue', '#060482', '#A80'
comm	A trait whose value must be an instance of a specified class.
cross_validation_lock	A contextmanager for running a block with our cross validation lock set to True.
keys	An instance of a Python list.
log	A trait whose value must be an instance of a specified class.
marks	An instance of a Python list.
model_id	Gets the model id of this widget.
selected	
selected_x	A numpy array trait type.
selected_y	A numpy array trait type.
types	
widget_types	
widgets	
x_scale	A trait whose value must be an instance of a specified class.
y_scale	A trait whose value must be an instance of a specified class.

bqplot.interacts.HandDraw

class bqplot.interacts.**HandDraw**(***kwargs*)

A hand-draw interaction.

This can be used to edit the 'y' value of an existing line using the mouse. The minimum and maximum x values of the line which can be edited may be passed as parameters. The y-values for any part of the line can be edited by drawing the desired path while holding the mouse-down. y-values corresponding to x-values smaller than

min_x or greater than max_x cannot be edited by HandDraw.

lines
The instance of Lines which is edited using the hand-draw interaction. The 'y' values of the line are changed according to the path of the mouse. If the lines has multi dimensional 'y', then the 'line_index' attribute is used to selected the 'y' to be edited.

Type an instance Lines mark or None (default: None)

line_index
For a line with multi-dimensional 'y', this indicates the index of the 'y' to be edited by the handdraw.

Type nonnegative integer (default: 0)

min_x
The minimum value of 'x' which should be edited via the handdraw.

Type float or *Date* or None (default: None)

max_x
The maximum value of 'x' which should be edited via the handdraw.

Type float or *Date* or None (default: None)

__init__ (***kwargs*)
Public constructor

Methods

__init__(**kwargs)	Public constructor
add_traits(**traits)	Dynamically add trait attributes to the Widget.
class_own_trait_events(name)	Get a dict of all event handlers defined on this class, not a parent.
class_own_traits(**metadata)	Get a dict of all the traitlets defined on this class, not a parent.
class_trait_names(**metadata)	Get a list of all the names of this class' traits.
class_traits(**metadata)	Get a dict of all the traits of this class.
close()	Close method.
close_all()	
get_manager_state([drop_defaults, widgets])	Returns the full state for a widget manager for embedding
get_state([key, drop_defaults])	Gets the widget state, or a piece of it.
get_view_spec()	
handle_comm_opened(comm, msg)	Static method, called when a widget is constructed.
has_trait(name)	Returns True if the object has a trait with the specified name.
hold_sync()	Hold syncing any state until the outermost context manager exits
hold_trait_notifications()	Context manager for bundling trait change notifications and cross validation.
notify_change(change)	Called when a property has changed.
observe(handler[, names, type])	Setup a handler to be called when a trait changes.
on_displayed(callback[, remove])	(Un)Register a widget displayed callback.
on_msg(callback[, remove])	(Un)Register a custom msg receive callback.

Continued on next page

Table 73 – continued from previous page

on_trait_change([handler, name, remove])	DEPRECATED: Setup a handler to be called when a trait changes.
on_widget_constructed(callback)	Registers a callback to be called when a widget is constructed.
open()	Open a comm to the frontend if one isn't already open.
send(content[, buffers])	Sends a custom msg to the widget model in the front-end.
send_state([key])	Sends the widget state, or a piece of it, to the front-end, if it exists.
set_state(sync_data)	Called when a state is received from the front-end.
set_trait(name, value)	Forcibly sets trait attribute, including read-only attributes.
setup_instance(*args, **kwargs)	This is called **before** self.__init__ is called.
trait_events([name])	Get a dict of all the event handlers of this class.
trait_metadata(traitname, key[, default])	Get metadata values for trait by key.
trait_names(**metadata)	Get a list of all the names of this class' traits.
traits(**metadata)	Get a dict of all the traits of this class.
unobserve(handler[, names, type])	Remove a trait change handler.
unobserve_all([name])	Remove trait change handlers of any type for the specified name.

Attributes

comm	A trait whose value must be an instance of a specified class.
cross_validation_lock	A contextmanager for running a block with our cross validation lock set to True.
keys	An instance of a Python list.
line_index	An int trait.
lines	A trait whose value must be an instance of a specified class.
log	A trait whose value must be an instance of a specified class.
max_x	A trait type representing a Union type.
min_x	A trait type representing a Union type.
model_id	Gets the model id of this widget.
types	
widget_types	
widgets	

bqplot.interacts.IndexSelector

class bqplot.interacts.**IndexSelector**(***kwargs*)

Index selector interaction.

This 1-D selector interaction uses the mouse x-coordinate to select the corresponding point in terms of the selector scale.

Index Selector has two modes:

1. default mode: The mouse controls the x-position of the selector.

2. **frozen mode: In this mode, the selector is frozen at a point and** does not respond to mouse events.

A single click switches between the two modes.

selected
A single element array containing the point corresponding the x-position of the mouse. This attribute is updated as you move the mouse along the x-direction on the figure.

Type numpy.ndarray

color
Color of the line representing the index selector.

Type Color or None (default: None)

line_width
Width of the line represetning the index selector.

Type nonnegative integer (default: 0)

__init__(***kwargs*)
Public constructor

Methods

__init__(**kwargs)	Public constructor
add_traits(**traits)	Dynamically add trait attributes to the Widget.
class_own_trait_events(name)	Get a dict of all event handlers defined on this class, not a parent.
class_own_traits(**metadata)	Get a dict of all the traitlets defined on this class, not a parent.
class_trait_names(**metadata)	Get a list of all the names of this class' traits.
class_traits(**metadata)	Get a dict of all the traits of this class.
close()	Close method.
close_all()	
get_manager_state([drop_defaults, widgets])	Returns the full state for a widget manager for embedding
get_state([key, drop_defaults])	Gets the widget state, or a piece of it.
get_view_spec()	
handle_comm_opened(comm, msg)	Static method, called when a widget is constructed.
has_trait(name)	Returns True if the object has a trait with the specified name.
hold_sync()	Hold syncing any state until the outermost context manager exits
hold_trait_notifications()	Context manager for bundling trait change notifications and cross validation.
notify_change(change)	Called when a property has changed.
observe(handler[, names, type])	Setup a handler to be called when a trait changes.
on_displayed(callback[, remove])	(Un)Register a widget displayed callback.
on_msg(callback[, remove])	(Un)Register a custom msg receive callback.
on_trait_change([handler, name, remove])	DEPRECATED: Setup a handler to be called when a trait changes.
on_widget_constructed(callback)	Registers a callback to be called when a widget is constructed.

Continued on next page

Table 75 – continued from previous page

open()	Open a comm to the frontend if one isn't already open.
reset()	
send(content[, buffers])	Sends a custom msg to the widget model in the front-end.
send_state([key])	Sends the widget state, or a piece of it, to the front-end, if it exists.
set_state(sync_data)	Called when a state is received from the front-end.
set_trait(name, value)	Forcibly sets trait attribute, including read-only attributes.
setup_instance(*args, **kwargs)	This is called **before** self.__init__ is called.
trait_events([name])	Get a dict of all the event handlers of this class.
trait_metadata(traitname, key[, default])	Get metadata values for trait by key.
trait_names(**metadata)	Get a list of all the names of this class' traits.
traits(**metadata)	Get a dict of all the traits of this class.
unobserve(handler[, names, type])	Remove a trait change handler.
unobserve_all([name])	Remove trait change handlers of any type for the specified name.

Attributes

color	A string holding a valid HTML color such as 'blue', '#060482', '#A80'
comm	A trait whose value must be an instance of a specified class.
cross_validation_lock	A contextmanager for running a block with our cross validation lock set to True.
keys	An instance of a Python list.
line_width	An int trait.
log	A trait whose value must be an instance of a specified class.
marks	An instance of a Python list.
model_id	Gets the model id of this widget.
scale	A trait whose value must be an instance of a specified class.
selected	A numpy array trait type.
types	
widget_types	
widgets	

bqplot.interacts.FastIntervalSelector

class bqplot.interacts.**FastIntervalSelector**(***kwargs*)

Fast interval selector interaction.

This 1-D selector is used to select an interval on the x-scale by just moving the mouse (without clicking or dragging). The x-coordinate of the mouse controls the mid point of the interval selected while the y-coordinate of the mouse controls the the width of the interval. The larger the y-coordinate, the wider the interval selected.

Interval selector has three modes:

1. **default mode: This is the default mode in which the mouse controls** the location and width of the

interval.

2. **fixed-width mode: In this mode the width of the interval is frozen** and only the location of the interval is controlled with the mouse. A single click from the default mode takes you to this mode. Another single click takes you back to the default mode.

3. **frozen mode: In this mode the selected interval is frozen and the** selector does not respond to mouse move. A double click from the default mode takes you to this mode. Another double click takes you back to the default mode.

selected
Two-element array containing the start and end of the interval selected in terms of the scale of the selector.

Type numpy.ndarray

color
color of the rectangle representing the interval selector

Type Color or None (default: None)

size
if not None, this is the fixed pixel-width of the interval selector

Type Float or None (default: None)

__init__(***kwargs*)
Public constructor

Methods

__init__(**kwargs)	Public constructor
add_traits(**traits)	Dynamically add trait attributes to the Widget.
class_own_trait_events(name)	Get a dict of all event handlers defined on this class, not a parent.
class_own_traits(**metadata)	Get a dict of all the traitlets defined on this class, not a parent.
class_trait_names(**metadata)	Get a list of all the names of this class' traits.
class_traits(**metadata)	Get a dict of all the traits of this class.
close()	Close method.
close_all()	
get_manager_state([drop_defaults, widgets])	Returns the full state for a widget manager for embedding
get_state([key, drop_defaults])	Gets the widget state, or a piece of it.
get_view_spec()	
handle_comm_opened(comm, msg)	Static method, called when a widget is constructed.
has_trait(name)	Returns True if the object has a trait with the specified name.
hold_sync()	Hold syncing any state until the outermost context manager exits
hold_trait_notifications()	Context manager for bundling trait change notifications and cross validation.
notify_change(change)	Called when a property has changed.
observe(handler[, names, type])	Setup a handler to be called when a trait changes.
on_displayed(callback[, remove])	(Un)Register a widget displayed callback.
on_msg(callback[, remove])	(Un)Register a custom msg receive callback.

Continued on next page

Table 77 – continued from previous page

on_trait_change([handler, name, remove])	DEPRECATED: Setup a handler to be called when a trait changes.
on_widget_constructed(callback)	Registers a callback to be called when a widget is constructed.
open()	Open a comm to the frontend if one isn't already open.
reset()	
send(content[, buffers])	Sends a custom msg to the widget model in the front-end.
send_state([key])	Sends the widget state, or a piece of it, to the front-end, if it exists.
set_state(sync_data)	Called when a state is received from the front-end.
set_trait(name, value)	Forcibly sets trait attribute, including read-only attributes.
setup_instance(*args, **kwargs)	This is called **before** self.__init__ is called.
trait_events([name])	Get a dict of all the event handlers of this class.
trait_metadata(traitname, key[, default])	Get metadata values for trait by key.
trait_names(**metadata)	Get a list of all the names of this class' traits.
traits(**metadata)	Get a dict of all the traits of this class.
unobserve(handler[, names, type])	Remove a trait change handler.
unobserve_all([name])	Remove trait change handlers of any type for the specified name.

Attributes

color	A string holding a valid HTML color such as 'blue', '#060482', '#A80'
comm	A trait whose value must be an instance of a specified class.
cross_validation_lock	A contextmanager for running a block with our cross validation lock set to True.
keys	An instance of a Python list.
log	A trait whose value must be an instance of a specified class.
marks	An instance of a Python list.
model_id	Gets the model id of this widget.
scale	A trait whose value must be an instance of a specified class.
selected	A numpy array trait type.
size	A float trait.
types	
widget_types	
widgets	

bqplot.interacts.MultiSelector

class bqplot.interacts.**MultiSelector**(***kwargs*)

Multi selector interaction.

This 1-D selector interaction enables the user to select multiple intervals using the mouse. A mouse-down marks the start of the interval. The drag after the mouse down in the x-direction selects the extent and a mouse-up

signifies the end of the interval.

The current selector is highlighted with a green border and the inactive selectors are highlighted with a red border.

The multi selector has three modes:

1. **default mode: In this mode the interaction behaves exactly as the** brush selector interaction with the current selector.
2. **add mode: In this mode a new selector can be added by clicking at** a point and dragging over the interval of interest. Once a new selector has been added, the multi selector is back in the default mode. From the default mode, ctrl+click switches to the add mode.
3. **choose mode: In this mode, any of the existing inactive selectors** can be set as the active selector. When an inactive selector is selected by clicking, the multi selector goes back to the default mode. From the default mode, shift+click switches to the choose mode.

A double click at the same point without moving the mouse in the x-direction will result in the entire interval being selected for the current selector.

selected
A dictionary with keys being the names of the intervals and values being the two element arrays containing the start and end of the interval selected by that particular selector in terms of the scale of the selector. This is a read-only attribute. This attribute changes while the selection is being made with the MultiSelectorinteraction.

Type dict

brushing
A boolean attribute to indicate if the selector is being dragged. It is True when the selector is being moved and false when it is not. This attribute can be used to trigger computationally intensive code which should be run only on the interval selection being completed as opposed to code which should be run whenever selected is changing.

Type bool (default: False)

names
A list of strings indicating the keys of the different intervals. Default values are 'int1', 'int2', 'int3' and so on.

Type list

show_names
Attribute to indicate if the names of the intervals are to be displayed along with the interval.

Type bool (default: True)

__init__(***kwargs*)
Public constructor

Methods

__init__(**kwargs)	Public constructor
add_traits(**traits)	Dynamically add trait attributes to the Widget.
class_own_trait_events(name)	Get a dict of all event handlers defined on this class, not a parent.
class_own_traits(**metadata)	Get a dict of all the traitlets defined on this class, not a parent.

Continued on next page

Table 79 – continued from previous page

class_trait_names(**metadata)	Get a list of all the names of this class' traits.
class_traits(**metadata)	Get a dict of all the traits of this class.
close()	Close method.
close_all()	
get_manager_state([drop_defaults, widgets])	Returns the full state for a widget manager for embedding
get_state([key, drop_defaults])	Gets the widget state, or a piece of it.
get_view_spec()	
handle_comm_opened(comm, msg)	Static method, called when a widget is constructed.
has_trait(name)	Returns True if the object has a trait with the specified name.
hidden_selected_changed(name, selected)	
hold_sync()	Hold syncing any state until the outermost context manager exits
hold_trait_notifications()	Context manager for bundling trait change notifications and cross validation.
notify_change(change)	Called when a property has changed.
observe(handler[, names, type])	Setup a handler to be called when a trait changes.
on_displayed(callback[, remove])	(Un)Register a widget displayed callback.
on_msg(callback[, remove])	(Un)Register a custom msg receive callback.
on_trait_change([handler, name, remove])	DEPRECATED: Setup a handler to be called when a trait changes.
on_widget_constructed(callback)	Registers a callback to be called when a widget is constructed.
open()	Open a comm to the frontend if one isn't already open.
reset()	
send(content[, buffers])	Sends a custom msg to the widget model in the front-end.
send_state([key])	Sends the widget state, or a piece of it, to the front-end, if it exists.
set_state(sync_data)	Called when a state is received from the front-end.
set_trait(name, value)	Forcibly sets trait attribute, including read-only attributes.
setup_instance(*args, **kwargs)	This is called **before** self.__init__ is called.
trait_events([name])	Get a dict of all the event handlers of this class.
trait_metadata(traitname, key[, default])	Get metadata values for trait by key.
trait_names(**metadata)	Get a list of all the names of this class' traits.
traits(**metadata)	Get a dict of all the traits of this class.
unobserve(handler[, names, type])	Remove a trait change handler.
unobserve_all([name])	Remove trait change handlers of any type for the specified name.

Attributes

brushing	A boolean (True, False) trait.
color	A string holding a valid HTML color such as 'blue', '#060482', '#A80'
comm	A trait whose value must be an instance of a specified class.

Continued on next page

Table 80 – continued from previous page

cross_validation_lock	A contextmanager for running a block with our cross validation lock set to True.
keys	An instance of a Python list.
log	A trait whose value must be an instance of a specified class.
marks	An instance of a Python list.
model_id	Gets the model id of this widget.
names	An instance of a Python list.
orientation	An enum whose value must be in a given sequence.
scale	A trait whose value must be an instance of a specified class.
selected	An instance of a Python dict.
show_names	A boolean (True, False) trait.
types	
widget_types	
widgets	

bqplot.interacts.OneDSelector

class bqplot.interacts.**OneDSelector**(***kwargs*)

One-dimensional selector interaction

Base class for all selectors which select data in one dimension, i.e., either the x or the y direction. The scale attribute should be provided.

scale

This is the scale which is used for inversion from the pixels to data co-ordinates. This scale is used for setting the selected attribute for the selector.

Type An instance of Scale

__init__(***kwargs*)

Public constructor

Methods

__init__(**kwargs)	Public constructor
add_traits(**traits)	Dynamically add trait attributes to the Widget.
class_own_trait_events(name)	Get a dict of all event handlers defined on this class, not a parent.
class_own_traits(**metadata)	Get a dict of all the traitlets defined on this class, not a parent.
class_trait_names(**metadata)	Get a list of all the names of this class' traits.
class_traits(**metadata)	Get a dict of all the traits of this class.
close()	Close method.
close_all()	
get_manager_state([drop_defaults, widgets])	Returns the full state for a widget manager for embedding
get_state([key, drop_defaults])	Gets the widget state, or a piece of it.
get_view_spec()	
handle_comm_opened(comm, msg)	Static method, called when a widget is constructed.

Continued on next page

Table 81 – continued from previous page

has_trait(name)	Returns True if the object has a trait with the specified name.
hold_sync()	Hold syncing any state until the outermost context manager exits
hold_trait_notifications()	Context manager for bundling trait change notifications and cross validation.
notify_change(change)	Called when a property has changed.
observe(handler[, names, type])	Setup a handler to be called when a trait changes.
on_displayed(callback[, remove])	(Un)Register a widget displayed callback.
on_msg(callback[, remove])	(Un)Register a custom msg receive callback.
on_trait_change([handler, name, remove])	DEPRECATED: Setup a handler to be called when a trait changes.
on_widget_constructed(callback)	Registers a callback to be called when a widget is constructed.
open()	Open a comm to the frontend if one isn't already open.
reset()	
send(content[, buffers])	Sends a custom msg to the widget model in the front-end.
send_state([key])	Sends the widget state, or a piece of it, to the front-end, if it exists.
set_state(sync_data)	Called when a state is received from the front-end.
set_trait(name, value)	Forcibly sets trait attribute, including read-only attributes.
setup_instance(*args, **kwargs)	This is called **before** self.__init__ is called.
trait_events([name])	Get a dict of all the event handlers of this class.
trait_metadata(traitname, key[, default])	Get metadata values for trait by key.
trait_names(**metadata)	Get a list of all the names of this class' traits.
traits(**metadata)	Get a dict of all the traits of this class.
unobserve(handler[, names, type])	Remove a trait change handler.
unobserve_all([name])	Remove trait change handlers of any type for the specified name.

Attributes

comm	A trait whose value must be an instance of a specified class.
cross_validation_lock	A contextmanager for running a block with our cross validation lock set to True.
keys	An instance of a Python list.
log	A trait whose value must be an instance of a specified class.
marks	An instance of a Python list.
model_id	Gets the model id of this widget.
scale	A trait whose value must be an instance of a specified class.
types	
widget_types	
widgets	

bqplot.interacts.Interaction

class bqplot.interacts.**Interaction**(***kwargs*)

The base interaction class.

An interaction is a mouse interaction layer for a figure that requires the capture of all mouse events on the plot area. A consequence is that one can allow only one interaction at any time on a figure.

An interaction can be associated with features such as selection or manual change of specific mark. Although, they differ from the so called 'mark interactions' in that they do not rely on knowing whether a specific element of the mark are hovered by the mouse.

types

A registry of existing interaction types.

Type dict (class-level attribute) representing interaction types

__init__(***kwargs*)

Public constructor

Methods

__init__(**kwargs)	Public constructor
add_traits(**traits)	Dynamically add trait attributes to the Widget.
class_own_trait_events(name)	Get a dict of all event handlers defined on this class, not a parent.
class_own_traits(**metadata)	Get a dict of all the traitlets defined on this class, not a parent.
class_trait_names(**metadata)	Get a list of all the names of this class' traits.
class_traits(**metadata)	Get a dict of all the traits of this class.
close()	Close method.
close_all()	
get_manager_state([drop_defaults, widgets])	Returns the full state for a widget manager for embedding
get_state([key, drop_defaults])	Gets the widget state, or a piece of it.
get_view_spec()	
handle_comm_opened(comm, msg)	Static method, called when a widget is constructed.
has_trait(name)	Returns True if the object has a trait with the specified name.
hold_sync()	Hold syncing any state until the outermost context manager exits
hold_trait_notifications()	Context manager for bundling trait change notifications and cross validation.
notify_change(change)	Called when a property has changed.
observe(handler[, names, type])	Setup a handler to be called when a trait changes.
on_displayed(callback[, remove])	(Un)Register a widget displayed callback.
on_msg(callback[, remove])	(Un)Register a custom msg receive callback.
on_trait_change([handler, name, remove])	DEPRECATED: Setup a handler to be called when a trait changes.
on_widget_constructed(callback)	Registers a callback to be called when a widget is constructed.
open()	Open a comm to the frontend if one isn't already open.

Continued on next page

Table 83 – continued from previous page

send(content[, buffers])	Sends a custom msg to the widget model in the front-end.
send_state([key])	Sends the widget state, or a piece of it, to the front-end, if it exists.
set_state(sync_data)	Called when a state is received from the front-end.
set_trait(name, value)	Forcibly sets trait attribute, including read-only attributes.
setup_instance(*args, **kwargs)	This is called **before** self.__init__ is called.
trait_events([name])	Get a dict of all the event handlers of this class.
trait_metadata(traitname, key[, default])	Get metadata values for trait by key.
trait_names(**metadata)	Get a list of all the names of this class' traits.
traits(**metadata)	Get a dict of all the traits of this class.
unobserve(handler[, names, type])	Remove a trait change handler.
unobserve_all([name])	Remove trait change handlers of any type for the specified name.

Attributes

comm	A trait whose value must be an instance of a specified class.
cross_validation_lock	A contextmanager for running a block with our cross validation lock set to True.
keys	An instance of a Python list.
log	A trait whose value must be an instance of a specified class.
model_id	Gets the model id of this widget.
types	
widget_types	
widgets	

bqplot.interacts.PanZoom

class bqplot.interacts.**PanZoom**(***kwargs*)

An interaction to pan and zoom wrt scales.

allow_pan

Toggle the ability to pan.

Type bool (default: True)

allow_zoom

Toggle the ability to zoom.

Type bool (default: True)

scales

Dictionary with keys such as 'x' and 'y' and values being the scales in the corresponding direction (dimensions) which should be panned or zoomed.

Type Dictionary of lists of Scales (default: {})

__init__(***kwargs*)

Public constructor

Methods

__init__(**kwargs)	Public constructor
add_traits(**traits)	Dynamically add trait attributes to the Widget.
class_own_trait_events(name)	Get a dict of all event handlers defined on this class, not a parent.
class_own_traits(**metadata)	Get a dict of all the traitlets defined on this class, not a parent.
class_trait_names(**metadata)	Get a list of all the names of this class' traits.
class_traits(**metadata)	Get a dict of all the traits of this class.
close()	Close method.
close_all()	
get_manager_state([drop_defaults, widgets])	Returns the full state for a widget manager for embedding
get_state([key, drop_defaults])	Gets the widget state, or a piece of it.
get_view_spec()	
handle_comm_opened(comm, msg)	Static method, called when a widget is constructed.
has_trait(name)	Returns True if the object has a trait with the specified name.
hold_sync()	Hold syncing any state until the outermost context manager exits
hold_trait_notifications()	Context manager for bundling trait change notifications and cross validation.
notify_change(change)	Called when a property has changed.
observe(handler[, names, type])	Setup a handler to be called when a trait changes.
on_displayed(callback[, remove])	(Un)Register a widget displayed callback.
on_msg(callback[, remove])	(Un)Register a custom msg receive callback.
on_trait_change([handler, name, remove])	DEPRECATED: Setup a handler to be called when a trait changes.
on_widget_constructed(callback)	Registers a callback to be called when a widget is constructed.
open()	Open a comm to the frontend if one isn't already open.
send(content[, buffers])	Sends a custom msg to the widget model in the front-end.
send_state([key])	Sends the widget state, or a piece of it, to the front-end, if it exists.
set_state(sync_data)	Called when a state is received from the front-end.
set_trait(name, value)	Forcibly sets trait attribute, including read-only attributes.
setup_instance(*args, **kwargs)	This is called **before** self.__init__ is called.
trait_events([name])	Get a dict of all the event handlers of this class.
trait_metadata(traitname, key[, default])	Get metadata values for trait by key.
trait_names(**metadata)	Get a list of all the names of this class' traits.
traits(**metadata)	Get a dict of all the traits of this class.
unobserve(handler[, names, type])	Remove a trait change handler.
unobserve_all([name])	Remove trait change handlers of any type for the specified name.

Attributes

allow_pan	A boolean (True, False) trait.
allow_zoom	A boolean (True, False) trait.
comm	A trait whose value must be an instance of a specified class.
cross_validation_lock	A contextmanager for running a block with our cross validation lock set to True.
keys	An instance of a Python list.
log	A trait whose value must be an instance of a specified class.
model_id	Gets the model id of this widget.
scales	An instance of a Python dict.
types	
widget_types	
widgets	

bqplot.interacts.Selector

class bqplot.interacts.**Selector**(***kwargs*)

Selector interaction. A selector can be used to select a subset of data

Base class for all the selectors.

marks

list of marks for which the *selected* attribute is updated based on the data selected by the selector.

Type list (default: [])

__init__(***kwargs*)

Public constructor

Methods

__init__(**kwargs)	Public constructor
add_traits(**traits)	Dynamically add trait attributes to the Widget.
class_own_trait_events(name)	Get a dict of all event handlers defined on this class, not a parent.
class_own_traits(**metadata)	Get a dict of all the traitlets defined on this class, not a parent.
class_trait_names(**metadata)	Get a list of all the names of this class' traits.
class_traits(**metadata)	Get a dict of all the traits of this class.
close()	Close method.
close_all()	
get_manager_state([drop_defaults, widgets])	Returns the full state for a widget manager for embedding
get_state([key, drop_defaults])	Gets the widget state, or a piece of it.
get_view_spec()	
handle_comm_opened(comm, msg)	Static method, called when a widget is constructed.
has_trait(name)	Returns True if the object has a trait with the specified name.
hold_sync()	Hold syncing any state until the outermost context manager exits

Continued on next page

Table 87 – continued from previous page

hold_trait_notifications()	Context manager for bundling trait change notifications and cross validation.
notify_change(change)	Called when a property has changed.
observe(handler[, names, type])	Setup a handler to be called when a trait changes.
on_displayed(callback[, remove])	(Un)Register a widget displayed callback.
on_msg(callback[, remove])	(Un)Register a custom msg receive callback.
on_trait_change([handler, name, remove])	DEPRECATED: Setup a handler to be called when a trait changes.
on_widget_constructed(callback)	Registers a callback to be called when a widget is constructed.
open()	Open a comm to the frontend if one isn't already open.
reset()	
send(content[, buffers])	Sends a custom msg to the widget model in the front-end.
send_state([key])	Sends the widget state, or a piece of it, to the front-end, if it exists.
set_state(sync_data)	Called when a state is received from the front-end.
set_trait(name, value)	Forcibly sets trait attribute, including read-only attributes.
setup_instance(*args, **kwargs)	This is called **before** self.__init__ is called.
trait_events([name])	Get a dict of all the event handlers of this class.
trait_metadata(traitname, key[, default])	Get metadata values for trait by key.
trait_names(**metadata)	Get a list of all the names of this class' traits.
traits(**metadata)	Get a dict of all the traits of this class.
unobserve(handler[, names, type])	Remove a trait change handler.
unobserve_all([name])	Remove trait change handlers of any type for the specified name.

Attributes

comm	A trait whose value must be an instance of a specified class.
cross_validation_lock	A contextmanager for running a block with our cross validation lock set to True.
keys	An instance of a Python list.
log	A trait whose value must be an instance of a specified class.
marks	An instance of a Python list.
model_id	Gets the model id of this widget.
types	
widget_types	
widgets	

bqplot.interacts.TwoDSelector

class bqplot.interacts.**TwoDSelector**(***kwargs*)

Two-dimensional selector interaction.

Base class for all selectors which select data in both the x and y dimensions. The attributes 'x_scale' and 'y_scale' should be provided.

x_scale
This is the scale which is used for inversion from the pixels to data co-ordinates in the x-direction. This scale is used for setting the selected attribute for the selector along with `y_scale`.

Type An instance of Scale

y_scale
This is the scale which is used for inversion from the pixels to data co-ordinates in the y-direction. This scale is used for setting the selected attribute for the selector along with `x_scale`.

Type An instance of Scale

__init__(***kwargs*)
Public constructor

Methods

__init__(**kwargs)	Public constructor
add_traits(**traits)	Dynamically add trait attributes to the Widget.
class_own_trait_events(name)	Get a dict of all event handlers defined on this class, not a parent.
class_own_traits(**metadata)	Get a dict of all the traitlets defined on this class, not a parent.
class_trait_names(**metadata)	Get a list of all the names of this class' traits.
class_traits(**metadata)	Get a dict of all the traits of this class.
close()	Close method.
close_all()	
get_manager_state([drop_defaults, widgets])	Returns the full state for a widget manager for embedding
get_state([key, drop_defaults])	Gets the widget state, or a piece of it.
get_view_spec()	
handle_comm_opened(comm, msg)	Static method, called when a widget is constructed.
has_trait(name)	Returns True if the object has a trait with the specified name.
hold_sync()	Hold syncing any state until the outermost context manager exits
hold_trait_notifications()	Context manager for bundling trait change notifications and cross validation.
notify_change(change)	Called when a property has changed.
observe(handler[, names, type])	Setup a handler to be called when a trait changes.
on_displayed(callback[, remove])	(Un)Register a widget displayed callback.
on_msg(callback[, remove])	(Un)Register a custom msg receive callback.
on_trait_change([handler, name, remove])	DEPRECATED: Setup a handler to be called when a trait changes.
on_widget_constructed(callback)	Registers a callback to be called when a widget is constructed.
open()	Open a comm to the frontend if one isn't already open.
reset()	
send(content[, buffers])	Sends a custom msg to the widget model in the frontend.
send_state([key])	Sends the widget state, or a piece of it, to the frontend, if it exists.

Continued on next page

Table 89 – continued from previous page

set_state(sync_data)	Called when a state is received from the front-end.
set_trait(name, value)	Forcibly sets trait attribute, including read-only attributes.
setup_instance(*args, **kwargs)	This is called **before** self.__init__ is called.
trait_events([name])	Get a dict of all the event handlers of this class.
trait_metadata(traitname, key[, default])	Get metadata values for trait by key.
trait_names(**metadata)	Get a list of all the names of this class' traits.
traits(**metadata)	Get a dict of all the traits of this class.
unobserve(handler[, names, type])	Remove a trait change handler.
unobserve_all([name])	Remove trait change handlers of any type for the specified name.

Attributes

comm	A trait whose value must be an instance of a specified class.
cross_validation_lock	A contextmanager for running a block with our cross validation lock set to True.
keys	An instance of a Python list.
log	A trait whose value must be an instance of a specified class.
marks	An instance of a Python list.
model_id	Gets the model id of this widget.
types	
widget_types	
widgets	
x_scale	A trait whose value must be an instance of a specified class.
y_scale	A trait whose value must be an instance of a specified class.

3.1.7 Traits Types

Date([default_value])	A datetime trait type.

bqplot.traits.Date

class bqplot.traits.**Date**(*default_value=datetime.datetime(2018, 11, 8, 18, 50, 8, 802982)*, ***kwargs*)

A datetime trait type.

Converts the passed date into a string format that can be used to construct a JavaScript datetime.

__init__(*default_value=datetime.datetime(2018, 11, 8, 18, 50, 8, 802982)*, ***kwargs*)

Declare a traitlet.

If *allow_none* is True, None is a valid value in addition to any values that are normally valid. The default is up to the subclass. For most trait types, the default value for allow_none is False.

Extra metadata can be associated with the traitlet using the .tag() convenience method or by using the traitlet instance's .metadata dictionary.

Methods

__init__([default_value])	Declare a traitlet.
class_init(cls, name)	Part of the initialization which may depend on the underlying HasDescriptors class.
default_value_repr()	
error(obj, value)	
get(obj[, cls])	
get_default_value()	DEPRECATED: Retrieve the static default value for this trait.
get_metadata(key[, default])	DEPRECATED: Get a metadata value.
info()	
init_default_value(obj)	DEPRECATED: Set the static default value for the trait type.
instance_init(obj)	Part of the initialization which may depend on the underlying HasDescriptors instance.
set(obj, value)	
set_metadata(key, value)	DEPRECATED: Set a metadata key/value.
tag(**metadata)	Sets metadata and returns self.
validate(obj, value)	

Attributes

allow_none
default_value
info_text
metadata
name
read_only
this_class

3.1.8 Toolbar

Toolbar(**kwargs)	Default toolbar for bqplot figures.

bqplot.toolbar.Toolbar

class bqplot.toolbar.**Toolbar**(***kwargs*)

Default toolbar for bqplot figures.

The default toolbar provides three buttons:

- A *Panzoom* toggle button which enables panning and zooming the figure.
- A *Save* button to save the figure as a png image.
- A *Reset* button, which resets the figure position to its original state.

When the *Panzoom* button is toggled to True for the first time, a new instance of `PanZoom` widget is created. The created `PanZoom` widget uses the scales of all the marks that are on the figure at this point. When the *PanZoom* widget is toggled to False, the figure retrieves its previous interaction. When the *Reset* button is

pressed, the `PanZoom` widget is deleted and the figure scales reset to their initial state. We are back to the case where the PanZoom widget has never been set.

If new marks are added to the figure after the panzoom button is toggled, and these use new scales, those scales will not be panned or zoomed, unless the reset button is clicked.

figure
The figure to which the toolbar will apply.

Type instance of Figure

__init__(***kwargs*)
Public constructor

Methods

__init__(**kwargs)	Public constructor
add_class(className)	Adds a class to the top level element of the widget.
add_traits(**traits)	Dynamically add trait attributes to the Widget.
class_own_trait_events(name)	Get a dict of all event handlers defined on this class, not a parent.
class_own_traits(**metadata)	Get a dict of all the traitlets defined on this class, not a parent.
class_trait_names(**metadata)	Get a list of all the names of this class' traits.
class_traits(**metadata)	Get a dict of all the traits of this class.
close()	Close method.
close_all()	
get_manager_state([drop_defaults, widgets])	Returns the full state for a widget manager for embedding
get_state([key, drop_defaults])	Gets the widget state, or a piece of it.
get_view_spec()	
handle_comm_opened(comm, msg)	Static method, called when a widget is constructed.
has_trait(name)	Returns True if the object has a trait with the specified name.
hold_sync()	Hold syncing any state until the outermost context manager exits
hold_trait_notifications()	Context manager for bundling trait change notifications and cross validation.
notify_change(change)	Called when a property has changed.
observe(handler[, names, type])	Setup a handler to be called when a trait changes.
on_displayed(callback[, remove])	(Un)Register a widget displayed callback.
on_msg(callback[, remove])	(Un)Register a custom msg receive callback.
on_trait_change([handler, name, remove])	DEPRECATED: Setup a handler to be called when a trait changes.
on_widget_constructed(callback)	Registers a callback to be called when a widget is constructed.
open()	Open a comm to the frontend if one isn't already open.
remove_class(className)	Removes a class from the top level element of the widget.
send(content[, buffers])	Sends a custom msg to the widget model in the frontend.

Continued on next page

Table 95 – continued from previous page

send_state([key])	Sends the widget state, or a piece of it, to the front-end, if it exists.
set_state(sync_data)	Called when a state is received from the front-end.
set_trait(name, value)	Forcibly sets trait attribute, including read-only attributes.
setup_instance(*args, **kwargs)	This is called **before** self.__init__ is called.
trait_events([name])	Get a dict of all the event handlers of this class.
trait_metadata(traitname, key[, default])	Get metadata values for trait by key.
trait_names(**metadata)	Get a list of all the names of this class' traits.
traits(**metadata)	Get a dict of all the traits of this class.
unobserve(handler[, names, type])	Remove a trait change handler.
unobserve_all([name])	Remove trait change handlers of any type for the specified name.

Attributes

comm	A trait whose value must be an instance of a specified class.
cross_validation_lock	A contextmanager for running a block with our cross validation lock set to True.
figure	A trait whose value must be an instance of a specified class.
keys	An instance of a Python list.
layout	An instance trait which coerces a dict to an instance.
log	A trait whose value must be an instance of a specified class.
model_id	Gets the model id of this widget.
widget_types	
widgets	

3.1.9 Pyplot

figure([key, fig])	Creates figures and switches between figures.
show([key, display_toolbar])	Shows the current context figure in the output area.
axes([mark, options])	Draws axes corresponding to the scales of a given mark.
plot(*args, **kwargs)	Draw lines in the current context figure.
scatter(x, y, **kwargs)	Draw a scatter in the current context figure.
hist(sample[, options])	Draw a histogram in the current context figure.
bar(x, y, **kwargs)	Draws a bar chart in the current context figure.
ohlc(*args, **kwargs)	Draw OHLC bars or candle bars in the current context figure.
geo(map_data, **kwargs)	Draw a map in the current context figure.
clear()	Clears the current context figure of all marks axes and grid lines.
close(key)	Closes and unregister the context figure corresponding to the key.
current_figure()	Returns the current context figure.
scales([key, scales])	Creates and switches between context scales.
xlim(min, max)	Set the domain bounds of the current 'x' scale.

Continued on next page

Table 97 – continued from previous page

ylim(min, max)	Set the domain bounds of the current 'y' scale.
axes([mark, options])	Draws axes corresponding to the scales of a given mark.
xlabel([label, mark])	Sets the value of label for an axis whose associated scale has the dimension *x*.
ylabel([label, mark])	Sets the value of label for an axis whose associated scale has the dimension *y*.

bqplot.pyplot.figure

bqplot.pyplot.**figure**(*key=None*, *fig=None*, ***kwargs*)

Creates figures and switches between figures.

If a bqplot.Figure object is provided via the fig optional argument, this figure becomes the current context figure.

Otherwise:

- If no key is provided, a new empty context figure is created.
- If a key is provided for which a context already exists, the corresponding context becomes current.
- If a key is provided and no corresponding context exists, a new context is reated for that key and becomes current.

Besides, optional arguments allow to set or modify Attributes of the selected context figure.

Parameters

- **key** (*hashable, optional*) – Any variable that can be used as a key for a dictionary
- **fig** (*Figure, optional*) – A bqplot Figure

bqplot.pyplot.show

bqplot.pyplot.**show**(*key=None*, *display_toolbar=True*)

Shows the current context figure in the output area.

Parameters

- **key** (*hashable, optional*) – Any variable that can be used as a key for a dictionary.
- **display_toolbar** (*bool (default: True)*) – If True, a toolbar for different mouse interaction is displayed with the figure.

Raises KeyError – When no context figure is associated with the provided key.

Examples

```
>>> import numpy as np
>>> import pyplot as plt
>>> n = 100
>>> x = np.arange(n)
>>> y = np.cumsum(np.random.randn(n))
>>> plt.plot(x,y)
>>> plt.show()
```

bqplot.pyplot.axes

bqplot.pyplot.**axes**(*mark=None*, *options={}*, ***kwargs*)

Draws axes corresponding to the scales of a given mark.

It also returns a dictionary of drawn axes. If the mark is not provided, the last drawn mark is used.

Parameters

- **mark** (*Mark or None (default: None)*) – The mark to inspect to create axes. If None, the last mark drawn is used instead.
- **options** (*dict (default: {})*) – Options for the axes to be created. If a scale labeled 'x' is required for that mark, options['x'] contains optional keyword arguments for the constructor of the corresponding axis type.

bqplot.pyplot.plot

bqplot.pyplot.**plot**(**args*, ***kwargs*)

Draw lines in the current context figure.

Signature: *plot(x, y, **kwargs)* or *plot(y, **kwargs)*, depending of the length of the list of positional arguments. In the case where the *x* array is not provided.

Parameters

- **x** (*numpy.ndarray or list, 1d or 2d (optional)*) – The x-coordinates of the plotted line. When not provided, the function defaults to *numpy.arange(len(y))* x can be 1-dimensional or 2-dimensional.
- **y** (*numpy.ndarray or list, 1d or 2d*) – The y-coordinates of the plotted line. If argument *x* is 2-dimensional it must also be 2-dimensional.
- **marker_str** (*string*) – string representing line_style, marker and color. For e.g. 'g–o', 'sr' etc
- **options** (*dict (default: {})*) – Options for the scales to be created. If a scale labeled 'x' is required for that mark, options['x'] contains optional keyword arguments for the constructor of the corresponding scale type.
- **axes_options** (*dict (default: {})*) – Options for the axes to be created. If an axis labeled 'x' is required for that mark, axes_options['x'] contains optional keyword arguments for the constructor of the corresponding axis type.
- **figure** (*Figure or None*) – The figure to which the line is to be added. If the value is None, the current figure is used.

bqplot.pyplot.scatter

bqplot.pyplot.**scatter**(*x*, *y*, ***kwargs*)

Draw a scatter in the current context figure.

Parameters

- **x** (*numpy.ndarray, 1d*) – The x-coordinates of the data points.
- **y** (*numpy.ndarray, 1d*) – The y-coordinates of the data points.
- **options** (*dict (default: {})*) – Options for the scales to be created. If a scale labeled 'x' is required for that mark, options['x'] contains optional keyword arguments for the constructor of the corresponding scale type.

- **axes_options** (*dict (default: {})*) – Options for the axes to be created. If an axis labeled 'x' is required for that mark, axes_options['x'] contains optional keyword arguments for the constructor of the corresponding axis type.

bqplot.pyplot.hist

bqplot.pyplot.**hist**(*sample*, *options={}*, ***kwargs*)

Draw a histogram in the current context figure.

Parameters

- **sample** (*numpy.ndarray, 1d*) – The sample for which the histogram must be generated.
- **options** (*dict (default: {})*) – Options for the scales to be created. If a scale labeled 'counts' is required for that mark, options['counts'] contains optional keyword arguments for the constructor of the corresponding scale type.
- **axes_options** (*dict (default: {})*) – Options for the axes to be created. If an axis labeled 'counts' is required for that mark, axes_options['counts'] contains optional keyword arguments for the constructor of the corresponding axis type.

bqplot.pyplot.bar

bqplot.pyplot.**bar**(*x*, *y*, ***kwargs*)

Draws a bar chart in the current context figure.

Parameters

- **x** (*numpy.ndarray, 1d*) – The x-coordinates of the data points.
- **y** (*numpy.ndarray, 1d*) – The y-coordinates of the data pints.
- **options** (*dict (default: {})*) – Options for the scales to be created. If a scale labeled 'x' is required for that mark, options['x'] contains optional keyword arguments for the constructor of the corresponding scale type.
- **axes_options** (*dict (default: {})*) – Options for the axes to be created. If an axis labeled 'x' is required for that mark, axes_options['x'] contains optional keyword arguments for the constructor of the corresponding axis type.

bqplot.pyplot.ohlc

bqplot.pyplot.**ohlc**(**args*, ***kwargs*)

Draw OHLC bars or candle bars in the current context figure.

Signature: *ohlc(x, y, **kwargs)* or *ohlc(y, **kwargs)*, depending of the length of the list of positional arguments. In the case where the *x* array is not provided

Parameters

- **x** (*numpy.ndarray or list, 1d (optional)*) – The x-coordinates of the plotted line. When not provided, the function defaults to *numpy.arange(len(y))*.
- **y** (*numpy.ndarray or list, 2d*) – The ohlc (open/high/low/close) information. A two dimensional array. y must have the shape (n, 4).

- **options** (*dict (default: {})*) – Options for the scales to be created. If a scale labeled 'x' is required for that mark, options['x'] contains optional keyword arguments for the constructor of the corresponding scale type.
- **axes_options** (*dict (default: {})*) – Options for the axes to be created. If an axis labeled 'x' is required for that mark, axes_options['x'] contains optional keyword arguments for the constructor of the corresponding axis type.

bqplot.pyplot.geo

bqplot.pyplot.**geo**(*map_data*, ***kwargs*)

Draw a map in the current context figure.

Parameters

- **map_data** (*string or bqplot.map (default: WorldMap)*) – Name of the map or json file required for the map data.
- **options** (*dict (default: {})*) – Options for the scales to be created. If a scale labeled 'x' is required for that mark, options['x'] contains optional keyword arguments for the constructor of the corresponding scale type.
- **axes_options** (*dict (default: {})*) – Options for the axes to be created. If an axis labeled 'x' is required for that mark, axes_options['x'] contains optional keyword arguments for the constructor of the corresponding axis type.

bqplot.pyplot.clear

bqplot.pyplot.**clear**()

Clears the current context figure of all marks axes and grid lines.

bqplot.pyplot.close

bqplot.pyplot.**close**(*key*)

Closes and unregister the context figure corresponding to the key.

Parameters **key** (*hashable*) – Any variable that can be used as a key for a dictionary

bqplot.pyplot.current_figure

bqplot.pyplot.**current_figure**()

Returns the current context figure.

bqplot.pyplot.scales

bqplot.pyplot.**scales**(*key=None*, *scales={}*)

Creates and switches between context scales.

If no key is provided, a new blank context is created.

If a key is provided for which a context already exists, the existing context is set as the current context.

If a key is provided and no corresponding context exists, a new context is created for that key and set as the current context.

Parameters

- **key** (*hashable, optional*) – Any variable that can be used as a key for a dictionary
- **scales** (*dictionary*) – Dictionary of scales to be used in the new context

Example

```
>>> scales(scales={
>>>     'x': Keep,
>>>     'color': ColorScale(min=0, max=1)
>>> })
```

This creates a new scales context, where the 'x' scale is kept from the previous context, the 'color' scale is an instance of ColorScale provided by the user. Other scales, potentially needed such as the 'y' scale in the case of a line chart will be created on the fly when needed.

Notes

Every call to the function figure triggers a call to scales.

The *scales* parameter is ignored if the *key* argument is not Keep and context scales already exist for that key.

bqplot.pyplot.xlim

bqplot.pyplot.**xlim**(*min*, *max*)
Set the domain bounds of the current 'x' scale.

bqplot.pyplot.ylim

bqplot.pyplot.**ylim**(*min*, *max*)
Set the domain bounds of the current 'y' scale.

bqplot.pyplot.xlabel

bqplot.pyplot.**xlabel**(*label=None*, *mark=None*, ***kwargs*)
Sets the value of label for an axis whose associated scale has the dimension *x*.

Parameters **label** (*Unicode or None (default: None)*) – The label for x axis

bqplot.pyplot.ylabel

bqplot.pyplot.**ylabel**(*label=None*, *mark=None*, ***kwargs*)
Sets the value of label for an axis whose associated scale has the dimension *y*.

Parameters **label** (*Unicode or None (default: None)*) – The label for y axis

Python Module Index

b

bqplot, 5
bqplot.axes, 79
bqplot.figure, 5
bqplot.interacts, 91
bqplot.market_map, 84
bqplot.marks, 35
bqplot.pyplot, 115
bqplot.scales, 9
bqplot.toolbar, 113
bqplot.traits, 112

Index

Symbols

__init__() (bqplot.axes.Axis method), 80
__init__() (bqplot.axes.ColorAxis method), 82
__init__() (bqplot.figure.Figure method), 7
__init__() (bqplot.interacts.BrushIntervalSelector method), 92
__init__() (bqplot.interacts.BrushSelector method), 94
__init__() (bqplot.interacts.FastIntervalSelector method), 100
__init__() (bqplot.interacts.HandDraw method), 96
__init__() (bqplot.interacts.IndexSelector method), 98
__init__() (bqplot.interacts.Interaction method), 106
__init__() (bqplot.interacts.MultiSelector method), 102
__init__() (bqplot.interacts.OneDSelector method), 104
__init__() (bqplot.interacts.PanZoom method), 107
__init__() (bqplot.interacts.Selector method), 109
__init__() (bqplot.interacts.TwoDSelector method), 111
__init__() (bqplot.market_map.MarketMap method), 86
__init__() (bqplot.market_map.SquareMarketMap method), 88
__init__() (bqplot.marks.Bars method), 55
__init__() (bqplot.marks.FlexLine method), 44
__init__() (bqplot.marks.Graph method), 58
__init__() (bqplot.marks.GridHeatMap method), 61
__init__() (bqplot.marks.HeatMap method), 63
__init__() (bqplot.marks.Hist method), 51
__init__() (bqplot.marks.Label method), 66
__init__() (bqplot.marks.Lines method), 41
__init__() (bqplot.marks.Map method), 76
__init__() (bqplot.marks.Mark method), 37
__init__() (bqplot.marks.OHLC method), 70
__init__() (bqplot.marks.Pie method), 73
__init__() (bqplot.marks.Scatter method), 48
__init__() (bqplot.scales.AlbersUSA method), 30
__init__() (bqplot.scales.ColorScale method), 20
__init__() (bqplot.scales.DateColorScale method), 22
__init__() (bqplot.scales.DateScale method), 16
__init__() (bqplot.scales.GeoScale method), 26
__init__() (bqplot.scales.Gnomonic method), 32
__init__() (bqplot.scales.LinearScale method), 12
__init__() (bqplot.scales.LogScale method), 14
__init__() (bqplot.scales.Mercator method), 28
__init__() (bqplot.scales.OrdinalColorScale method), 24
__init__() (bqplot.scales.OrdinalScale method), 18
__init__() (bqplot.scales.Scale method), 10
__init__() (bqplot.scales.Stereographic method), 34
__init__() (bqplot.toolbar.Toolbar method), 114
__init__() (bqplot.traits.Date method), 112

A

AlbersUSA (class in bqplot.scales), 30
align (bqplot.marks.Bars attribute), 54
align (bqplot.marks.Label attribute), 66
allow_padding (bqplot.scales.Scale attribute), 10
allow_pan (bqplot.interacts.PanZoom attribute), 107
allow_zoom (bqplot.interacts.PanZoom attribute), 107
anchor_style (bqplot.marks.GridHeatMap attribute), 60
animation_duration (bqplot.figure.Figure attribute), 7
apply_clip (bqplot.marks.Mark attribute), 37
axes (bqplot.figure.Figure attribute), 6
axes (bqplot.market_map.MarketMap attribute), 85
axes() (in module bqplot.pyplot), 117
Axis (class in bqplot.axes), 79
axis_types (bqplot.axes.Axis attribute), 79

B

background_style (bqplot.figure.Figure attribute), 6
bar() (in module bqplot.pyplot), 118
Bars (class in bqplot.marks), 53
base (bqplot.marks.Bars attribute), 54
bins (bqplot.marks.Hist attribute), 51
bqplot (module), 5
bqplot.axes (module), 79
bqplot.figure (module), 5
bqplot.interacts (module), 91
bqplot.market_map (module), 84
bqplot.marks (module), 35
bqplot.pyplot (module), 115

bqplot.scales (module), 9
bqplot.toolbar (module), 113
bqplot.traits (module), 112
brushing (bqplot.interacts.BrushIntervalSelector attribute), 91
brushing (bqplot.interacts.BrushSelector attribute), 94
brushing (bqplot.interacts.MultiSelector attribute), 102
BrushIntervalSelector (class in bqplot.interacts), 91
BrushSelector (class in bqplot.interacts), 93

C

center (bqplot.scales.Gnomonic attribute), 32
center (bqplot.scales.Mercator attribute), 28
center (bqplot.scales.Stereographic attribute), 34
charge (bqplot.marks.Graph attribute), 57
clear() (in module bqplot.pyplot), 119
clip_angle (bqplot.scales.Gnomonic attribute), 32
clip_angle (bqplot.scales.Stereographic attribute), 34
close() (in module bqplot.pyplot), 119
close_path (bqplot.marks.Lines attribute), 40
color (bqplot.axes.Axis attribute), 80
color (bqplot.interacts.BrushIntervalSelector attribute), 91
color (bqplot.interacts.BrushSelector attribute), 94
color (bqplot.interacts.FastIntervalSelector attribute), 100
color (bqplot.interacts.IndexSelector attribute), 98
color (bqplot.market_map.MarketMap attribute), 84
color (bqplot.marks.Bars attribute), 55
color (bqplot.marks.FlexLine attribute), 44
color (bqplot.marks.Graph attribute), 58
color (bqplot.marks.GridHeatMap attribute), 61
color (bqplot.marks.HeatMap attribute), 63
color (bqplot.marks.Label attribute), 66
color (bqplot.marks.Lines attribute), 41
color (bqplot.marks.Map attribute), 76
color (bqplot.marks.Pie attribute), 73
color_mode (bqplot.marks.Bars attribute), 54
ColorAxis (class in bqplot.axes), 82
colors (bqplot.marks.Bars attribute), 54
colors (bqplot.marks.FlexLine attribute), 43
colors (bqplot.marks.Graph attribute), 58
colors (bqplot.marks.Hist attribute), 51
colors (bqplot.marks.Lines attribute), 40
colors (bqplot.marks.Map attribute), 76
colors (bqplot.marks.OHLC attribute), 69
colors (bqplot.marks.Pie attribute), 72
colors (bqplot.marks.Scatter attribute), 46
colors (bqplot.scales.ColorScale attribute), 20
ColorScale (class in bqplot.scales), 20
cols (bqplot.market_map.MarketMap attribute), 85
column (bqplot.marks.GridHeatMap attribute), 61
column_align (bqplot.marks.GridHeatMap attribute), 60
count (bqplot.marks.Hist attribute), 51
current_figure() (in module bqplot.pyplot), 119
curves_subset (bqplot.marks.Lines attribute), 40

D

Date (class in bqplot.traits), 112
DateColorScale (class in bqplot.scales), 22
DateScale (class in bqplot.scales), 16
default_colors (bqplot.marks.Scatter attribute), 46
default_opacities (bqplot.marks.Scatter attribute), 47
default_size (bqplot.marks.Label attribute), 65
default_size (bqplot.marks.Scatter attribute), 47
default_skew (bqplot.marks.Scatter attribute), 47
directed (bqplot.marks.Graph attribute), 57
display_labels (bqplot.marks.Pie attribute), 73
display_legend (bqplot.marks.Mark attribute), 36
display_name (bqplot.marks.Mark attribute), 36
display_names (bqplot.marks.Scatter attribute), 47
display_text (bqplot.market_map.MarketMap attribute), 84
display_values (bqplot.marks.Pie attribute), 73
domain (bqplot.scales.OrdinalColorScale attribute), 24
domain (bqplot.scales.OrdinalScale attribute), 18
domain_class (bqplot.scales.DateScale attribute), 16
domain_class (bqplot.scales.Scale attribute), 10
drag_size (bqplot.marks.Label attribute), 66
drag_size (bqplot.marks.Scatter attribute), 47
dtype (bqplot.scales.AlbersUSA attribute), 30
dtype (bqplot.scales.ColorScale attribute), 20
dtype (bqplot.scales.DateColorScale attribute), 22
dtype (bqplot.scales.DateScale attribute), 16
dtype (bqplot.scales.LinearScale attribute), 12
dtype (bqplot.scales.LogScale attribute), 14
dtype (bqplot.scales.Mercator attribute), 28
dtype (bqplot.scales.OrdinalColorScale attribute), 24
dtype (bqplot.scales.OrdinalScale attribute), 18

E

enable_hover (bqplot.marks.Mark attribute), 37
enable_move (bqplot.marks.Scatter attribute), 47
end_angle (bqplot.marks.Pie attribute), 73

F

FastIntervalSelector (class in bqplot.interacts), 99
fig_margin (bqplot.figure.Figure attribute), 7
figure (bqplot.toolbar.Toolbar attribute), 114
Figure (class in bqplot.figure), 6
figure() (in module bqplot.pyplot), 116
fill (bqplot.marks.Lines attribute), 40
fill_colors (bqplot.marks.Lines attribute), 40
fill_opacities (bqplot.marks.Lines attribute), 40
FlexLine (class in bqplot.marks), 43
font_size (bqplot.marks.Pie attribute), 73
font_weight (bqplot.marks.Label attribute), 66
font_weight (bqplot.marks.Pie attribute), 73
format (bqplot.marks.OHLC attribute), 69

G

geo() (in module bqplot.pyplot), 119
GeoScale (class in bqplot.scales), 26
Gnomonic (class in bqplot.scales), 31
Graph (class in bqplot.marks), 57
grid_color (bqplot.axes.Axis attribute), 80
grid_lines (bqplot.axes.Axis attribute), 80
GridHeatMap (class in bqplot.marks), 60
groups (bqplot.market_map.MarketMap attribute), 84

H

HandDraw (class in bqplot.interacts), 95
HeatMap (class in bqplot.marks), 63
highlight_links (bqplot.marks.Graph attribute), 57
Hist (class in bqplot.marks), 50
hist() (in module bqplot.pyplot), 118
hover_highlight (bqplot.marks.Map attribute), 76
hovered_styles (bqplot.marks.Map attribute), 76

I

icon (bqplot.axes.Axis attribute), 79
icon (bqplot.marks.Bars attribute), 54
icon (bqplot.marks.Hist attribute), 50
icon (bqplot.marks.Lines attribute), 39
icon (bqplot.marks.OHLC attribute), 69
icon (bqplot.marks.Scatter attribute), 46
IndexSelector (class in bqplot.interacts), 97
inner_radius (bqplot.marks.Pie attribute), 72
interaction (bqplot.figure.Figure attribute), 6
Interaction (class in bqplot.interacts), 106
interactions (bqplot.marks.Mark attribute), 37
interpolation (bqplot.marks.Lines attribute), 40

L

label (bqplot.axes.Axis attribute), 79
Label (class in bqplot.marks), 65
label_color (bqplot.axes.Axis attribute), 80
label_color (bqplot.marks.Pie attribute), 73
label_location (bqplot.axes.Axis attribute), 79
label_offset (bqplot.axes.Axis attribute), 80
labels (bqplot.marks.Mark attribute), 37
labels_visibility (bqplot.marks.Lines attribute), 40
legend_location (bqplot.figure.Figure attribute), 6
legend_style (bqplot.figure.Figure attribute), 6
legend_text (bqplot.figure.Figure attribute), 7
line_index (bqplot.interacts.HandDraw attribute), 96
line_style (bqplot.marks.Lines attribute), 40
line_width (bqplot.interacts.IndexSelector attribute), 98
LinearScale (class in bqplot.scales), 12
lines (bqplot.interacts.HandDraw attribute), 96
Lines (class in bqplot.marks), 39
link_color (bqplot.marks.Graph attribute), 58
link_data (bqplot.marks.Graph attribute), 57
link_distance (bqplot.marks.Graph attribute), 57
link_matrix (bqplot.marks.Graph attribute), 57
link_type (bqplot.marks.Graph attribute), 57
LogScale (class in bqplot.scales), 14

M

Map (class in bqplot.marks), 76
map_data (bqplot.marks.Map attribute), 76
map_margin (bqplot.market_map.MarketMap attribute), 86
Mark (class in bqplot.marks), 36
mark_types (bqplot.marks.Mark attribute), 36
marker (bqplot.marks.Lines attribute), 40
marker (bqplot.marks.OHLC attribute), 69
marker (bqplot.marks.Scatter attribute), 46
marker_size (bqplot.marks.Lines attribute), 40
MarketMap (class in bqplot.market_map), 84
marks (bqplot.figure.Figure attribute), 6
marks (bqplot.interacts.Selector attribute), 109
max (bqplot.scales.ColorScale attribute), 20
max (bqplot.scales.DateColorScale attribute), 22
max (bqplot.scales.DateScale attribute), 16
max (bqplot.scales.LinearScale attribute), 12
max (bqplot.scales.LogScale attribute), 14
max_aspect_ratio (bqplot.figure.Figure attribute), 7
max_aspect_ratio (bqplot.market_map.MarketMap attribute), 86
max_x (bqplot.interacts.HandDraw attribute), 96
Mercator (class in bqplot.scales), 28
mid (bqplot.scales.ColorScale attribute), 20
mid (bqplot.scales.DateColorScale attribute), 22
mid_range (bqplot.scales.LinearScale attribute), 12
midpoints (bqplot.marks.Hist attribute), 51
min (bqplot.scales.ColorScale attribute), 20
min (bqplot.scales.DateColorScale attribute), 22
min (bqplot.scales.DateScale attribute), 16
min (bqplot.scales.LinearScale attribute), 12
min (bqplot.scales.LogScale attribute), 14
min_aspect_ratio (bqplot.figure.Figure attribute), 7
min_aspect_ratio (bqplot.market_map.MarketMap attribute), 86
min_range (bqplot.scales.LinearScale attribute), 12
min_x (bqplot.interacts.HandDraw attribute), 96
MultiSelector (class in bqplot.interacts), 101

N

name (bqplot.marks.Bars attribute), 54
name (bqplot.marks.FlexLine attribute), 43
name (bqplot.marks.Hist attribute), 50
name (bqplot.marks.Lines attribute), 39
name (bqplot.marks.OHLC attribute), 69
name (bqplot.marks.Scatter attribute), 46
names (bqplot.interacts.MultiSelector attribute), 102
names (bqplot.market_map.MarketMap attribute), 84

names (bqplot.marks.Scatter attribute), 47
node_data (bqplot.marks.Graph attribute), 57
normalized (bqplot.marks.Hist attribute), 51
num_ticks (bqplot.axes.Axis attribute), 79

O

offset (bqplot.axes.Axis attribute), 79
OHLC (class in bqplot.marks), 69
ohlc() (in module bqplot.pyplot), 118
on_hover (bqplot.market_map.MarketMap attribute), 85
OneDSelector (class in bqplot.interacts), 104
opacities (bqplot.marks.Bars attribute), 54
opacities (bqplot.marks.Hist attribute), 51
opacities (bqplot.marks.Lines attribute), 40
opacities (bqplot.marks.OHLC attribute), 69
opacities (bqplot.marks.Pie attribute), 72
opacity (bqplot.marks.Label attribute), 66
OrdinalColorScale (class in bqplot.scales), 24
OrdinalScale (class in bqplot.scales), 18
orientation (bqplot.axes.Axis attribute), 79
orientation (bqplot.interacts.BrushIntervalSelector attribute), 91
orientation (bqplot.marks.Bars attribute), 54

P

padding (bqplot.marks.Bars attribute), 54
padding_x (bqplot.figure.Figure attribute), 6
padding_y (bqplot.figure.Figure attribute), 6
PanZoom (class in bqplot.interacts), 107
Pie (class in bqplot.marks), 72
plot() (in module bqplot.pyplot), 117
precedence (bqplot.scales.LinearScale attribute), 12
precedence (bqplot.scales.Scale attribute), 10
precision (bqplot.scales.Gnomonic attribute), 32
precision (bqplot.scales.Stereographic attribute), 34
preserve_domain (bqplot.marks.Mark attribute), 36

R

radius (bqplot.marks.Pie attribute), 72
ref_data (bqplot.market_map.MarketMap attribute), 84
restrict_x (bqplot.marks.Label attribute), 66
restrict_x (bqplot.marks.Scatter attribute), 47
restrict_y (bqplot.marks.Label attribute), 66
restrict_y (bqplot.marks.Scatter attribute), 47
reverse (bqplot.scales.Scale attribute), 10
rotate (bqplot.scales.Mercator attribute), 28
rotate (bqplot.scales.Stereographic attribute), 33
rotation (bqplot.marks.Label attribute), 66
row (bqplot.marks.GridHeatMap attribute), 61
row_align (bqplot.marks.GridHeatMap attribute), 60
row_groups (bqplot.market_map.MarketMap attribute), 85
rows (bqplot.market_map.MarketMap attribute), 85
rtype (bqplot.scales.AlbersUSA attribute), 30
rtype (bqplot.scales.ColorScale attribute), 20
rtype (bqplot.scales.DateColorScale attribute), 22
rtype (bqplot.scales.DateScale attribute), 16
rtype (bqplot.scales.LinearScale attribute), 12
rtype (bqplot.scales.LogScale attribute), 14
rtype (bqplot.scales.Mercator attribute), 28
rtype (bqplot.scales.OrdinalColorScale attribute), 24
rtype (bqplot.scales.OrdinalScale attribute), 18

S

sample (bqplot.marks.Hist attribute), 51
scale (bqplot.axes.Axis attribute), 79
scale (bqplot.axes.ColorAxis attribute), 82
scale (bqplot.interacts.OneDSelector attribute), 104
Scale (class in bqplot.scales), 10
scale_factor (bqplot.scales.AlbersUSA attribute), 30
scale_factor (bqplot.scales.Gnomonic attribute), 31
scale_factor (bqplot.scales.Mercator attribute), 28
scale_factor (bqplot.scales.Stereographic attribute), 33
scale_type (bqplot.scales.ColorScale attribute), 20
scale_types (bqplot.scales.Scale attribute), 10
scale_x (bqplot.figure.Figure attribute), 6
scale_y (bqplot.figure.Figure attribute), 6
scales (bqplot.interacts.PanZoom attribute), 107
scales (bqplot.market_map.MarketMap attribute), 85
scales (bqplot.marks.Mark attribute), 36
scales() (in module bqplot.pyplot), 119
scales_metadata (bqplot.marks.Mark attribute), 36
Scatter (class in bqplot.marks), 46
scatter() (in module bqplot.pyplot), 117
scheme (bqplot.scales.ColorScale attribute), 20
selected (bqplot.interacts.BrushIntervalSelector attribute), 91
selected (bqplot.interacts.BrushSelector attribute), 93
selected (bqplot.interacts.FastIntervalSelector attribute), 100
selected (bqplot.interacts.IndexSelector attribute), 98
selected (bqplot.interacts.MultiSelector attribute), 102
selected (bqplot.marks.Map attribute), 76
selected (bqplot.marks.Mark attribute), 37
selected_style (bqplot.marks.Mark attribute), 37
selected_styles (bqplot.marks.Map attribute), 76
selected_x (bqplot.interacts.BrushSelector attribute), 93
selected_y (bqplot.interacts.BrushSelector attribute), 93
Selector (class in bqplot.interacts), 109
show() (in module bqplot.pyplot), 116
show_groups (bqplot.market_map.MarketMap attribute), 85
show_names (bqplot.interacts.MultiSelector attribute), 102
side (bqplot.axes.Axis attribute), 79
size (bqplot.interacts.FastIntervalSelector attribute), 100
size (bqplot.marks.Label attribute), 66
sizes (bqplot.marks.Pie attribute), 73

sort (bqplot.marks.Pie attribute), 72
SquareMarketMap (class in bqplot.market_map), 88
stabilized (bqplot.scales.LinearScale attribute), 12
start_angle (bqplot.marks.Pie attribute), 73
Stereographic (class in bqplot.scales), 33
stroke (bqplot.marks.Bars attribute), 54
stroke (bqplot.marks.Hist attribute), 51
stroke (bqplot.marks.OHLC attribute), 69
stroke (bqplot.marks.Pie attribute), 72
stroke (bqplot.marks.Scatter attribute), 46
stroke_width (bqplot.marks.FlexLine attribute), 43
stroke_width (bqplot.marks.Lines attribute), 40
stroke_width (bqplot.marks.OHLC attribute), 69
stroke_width (bqplot.marks.Scatter attribute), 46

T

text (bqplot.marks.Label attribute), 65
tick_format (bqplot.axes.Axis attribute), 79
tick_rotate (bqplot.axes.Axis attribute), 80
tick_style (bqplot.axes.Axis attribute), 80
tick_values (bqplot.axes.Axis attribute), 79
title (bqplot.figure.Figure attribute), 6
title_style (bqplot.figure.Figure attribute), 7
Toolbar (class in bqplot.toolbar), 113
tooltip (bqplot.marks.Mark attribute), 37
tooltip_fields (bqplot.market_map.MarketMap attribute), 85
tooltip_formats (bqplot.market_map.MarketMap attribute), 85
tooltip_location (bqplot.marks.Mark attribute), 37
tooltip_style (bqplot.marks.Mark attribute), 37
tooltip_widget (bqplot.market_map.MarketMap attribute), 85
translate (bqplot.scales.AlbersUSA attribute), 30
TwoDSelector (class in bqplot.interacts), 110
type (bqplot.marks.Bars attribute), 54
types (bqplot.interacts.Interaction attribute), 106

U

unselected_style (bqplot.marks.Mark attribute), 37

V

values_format (bqplot.marks.Pie attribute), 73
visible (bqplot.axes.Axis attribute), 80
visible (bqplot.marks.Mark attribute), 37

W

width (bqplot.marks.FlexLine attribute), 44

X

x (bqplot.marks.Bars attribute), 54
x (bqplot.marks.FlexLine attribute), 44
x (bqplot.marks.Graph attribute), 58
x (bqplot.marks.HeatMap attribute), 63
x (bqplot.marks.Label attribute), 66
x (bqplot.marks.Lines attribute), 41
x (bqplot.marks.OHLC attribute), 70
x (bqplot.marks.Pie attribute), 72
x_offset (bqplot.marks.Label attribute), 65
x_scale (bqplot.interacts.TwoDSelector attribute), 110
xlabel() (in module bqplot.pyplot), 120
xlim() (in module bqplot.pyplot), 120

Y

y (bqplot.marks.Bars attribute), 55
y (bqplot.marks.FlexLine attribute), 44
y (bqplot.marks.Graph attribute), 58
y (bqplot.marks.HeatMap attribute), 63
y (bqplot.marks.Label attribute), 66
y (bqplot.marks.Lines attribute), 41
y (bqplot.marks.OHLC attribute), 70
y (bqplot.marks.Pie attribute), 72
y_offset (bqplot.marks.Label attribute), 65
y_scale (bqplot.interacts.TwoDSelector attribute), 111
ylabel() (in module bqplot.pyplot), 120
ylim() (in module bqplot.pyplot), 120